

# CI Pathway: Parallel Computing

## Assignment - 1

UCLA, Statistics  
Hochan Son  
Summer 2025  
June 25, 2025

## 1 Introduction

This analysis evaluates parallel computing performance using the Laplace equation solver as a benchmark. The study compares four implementation approaches across varying thread/process counts to assess scalability and efficiency characteristics.

## 2 Hardware Environment

### 2.1 System Specifications

Table 1: NCSA Delta Compute Environment

Component	Specification
Compute Platform	NCSA Delta HPC Cluster
Login Node	dt-login04.delta.ncsa.illinois.edu
Compute Node	cn034.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.

Our first exercises will be to compile, run and time the Laplace code using the two programming models in this Pathway: **OpenMP** and **MPI**. This will get you familiar with the programming environment in preparation for our actual parallel programming in the next two modules. It will also allow you to experience some parallel scaling first hand. Specifically, our goal will be to compile the Laplace code in OpenMP and run it on varying numbers of **threads** to see how much it speeds up. We will do the same thing using MPI to run with multiple **processes**. Don't worry if it seems like we are skipping over some details today - we are. But those details will be made clear in our following modules.

### 3.1 OpenMP

1. Go into the OpenMP folder inside the Exercises folder. You will see codes there called `laplace_omp.c` and `laplace_omp.f90`. Select whichever language most interests you.

To compile with OpenMP we do either:

```
nvc -mp laplace_omp.c
or
nvfortran -mp laplace_omp.f90
```

Now we have an executable called `a.out`. But we need to ask for a compute node with multiple cores allocated in order to run. The Slurm command to get us a command line (`-pty bash`) on a compute node (`--nodes=1`) with 32 cores (`--cpus-per-task=32`) is:

```

srun --account=becs-delta-cpu --partition=cpu-interactive \
    --nodes=1 --cpus-per-task=32 --pty bash

```

2. You will notice a few messages as Slurm find the resources, and then your command line will change to something with a compute node number in it. From the command line on this compute node we can run up to 32 cores. OpenMP allows us to control core usage with the `OMP_NUM_THREADS` environment variable. You learned about these in the Intro to Delta module. Set this variable to request 1 core ‘( `export ‘OMP_NUM_THREADS=1` ) and run with `a.out` to find the baseline run time for the Laplace code. If you select 4000 iterations, the code will run to a complete solution. It reports its own runtime for you. Now, try varying number of cores up to 32 and see what kind of speedup you experience. Note that you do not need to recompile the code. Just change the environment variable and run `a.out`. Record these times for our discussion. exit the compute node when you are finished. You should find yourself returned to the login node.

## 3.2 MPI

1. Now we will do a similar exercise using MPI. Go into the MPI folder inside the Exercises folder. You will see codes there called `laplace_mpi.c` and `laplace_mpi.f`. Select whichever language most interests you.

To compile with MPI we do either:

```

mpicc laplace_mpi.c
or
mpif90 laplace_mpi.f90

```

Again, you have an executable called `a.out`. Now you need to ask for a compute node with multiple processes allocated in order to run. Similar, but not identical, to the previous Slurm command, the one to get us a command line on a compute node with 4 processes (`--nodes=1 --tasks=4 --tasks-per-node=4`) is:

```

srun --account=becs-delta-cpu --partition=cpu-interactive \
    --nodes=1 --tasks=4 --tasks-per-node=4 --pty bash

```

2. With MPI we will limit ourselves to a single timing run, using 4 processes. The command to run our `a.out` executable on our four available processes is

```

mpirun -n 4 a.out

```

Record this time for our later discussion. Exit the compute node when you are finished.

## 4 Exercise

1. Exercise 1: Compile, run, and time the Laplace code using the two programming models in this Pathway: OpenMP and MPI. Vary the number of cores up to 32 by changing the `OMP_NUM_THREADS` environment variable to see what speedup you experience. Record these times.
2. Exercise 2: Execute a single timing run, using 4 processes, and record the time obtained.

## 5 Solutions: OpenMP and MPI Performance Analysis

### 5.1 Serial, OpenMP, and Enhanced Parallel Processing

The purpose of this exercise is to measure the differences in how parallelism benefits the speed of the execution by distributed computation over multiple CPU cores. I've done a few observations. 1. Varying threads (1, 8, and 32) on `OMP_NUM_THREADS` counts. 2. Varying techniques such as serial, OpenMP, and red-black (checkerboard algorithm) by compiler tuning. The performance evaluation reveals significant differences across parallelization approaches, as summarized in the following tables.

Each of the processes has completed with 1, 8, and 32 threads. It has shown that the serial process has no performance improvement across thread counts. The OpenMP has shown good performance. Although Enhanced Parallel test with red-black checkerboard algorithm has the best outcome at threads=32. However, the efficiency on 32 threads has decreased. As thread count increases, efficiency typically decreases due to parallel overhead, communication costs, and workload imbalances.

Table 3: Parallel Processing Performance Comparison

Method	Threads	Time (s)	Speedup	Efficiency (%)	Iterations
Serial Process Test	1	22.040	1.00×	100.0	3372
Serial Process Test	8	22.037	1.00×	12.5	3372
Serial Process Test	32	22.065	1.00×	3.1	3372
OpenMP Process Test	1	21.733	1.00×	100.0	3372
OpenMP Process Test	8	4.175	5.21×	65.1	3372
OpenMP Process Test	32	1.992	10.91×	34.1	3372
Enhanced (Red/Black) Parallel Test	1	12.738	1.00×	100.0	3279
Enhanced (Red/Black) Parallel Test	8	2.569	4.96×	62.0	3279
Enhanced (Red/Black) Parallel Test	32	1.490	8.55×	26.7	3279

- **Serial Test:** Shows no parallelization benefit, confirming serial execution regardless of thread count.
- **OpenMP Implementation:**  
Demonstrates significant speedup with increasing thread count:

- 8 threads:  $5.21\times$  speedup (65.1% efficiency)
- 32 threads:  $10.91\times$  speedup (34.1% efficiency)
- **Enhanced Red/Black (checkerboard algorithm) Method:** Achieves best overall performance:
  - Superior single-thread performance (12.738s vs 21.733s)
  - Best 32-thread performance (1.490s vs 1.992s)
  - 25.2% faster than OpenMP at 32 threads
  - Slight reduction in required iterations (3279 vs 3372)
- **Efficiency Analysis:** Both parallel methods show decreasing efficiency with higher thread counts, typical of parallel overhead and diminishing returns.

## 5.2 Serial, MPI Processing

The MPI implementation of the Laplace code demonstrates a different approach to parallelization compared to OpenMP. The key observations from the MPI results are as follows:

Table 4: MPI vs Serial Process Performance Comparison

Test Type	Processes	Time (s)	Speedup	Efficiency (%)	Status
Serial Process Test	1	46.711	$1.00\times$	100.0	Success
	4	21.878	$2.14\times$	53.5	Partial*
MPI Test	1	–	–	–	Failed**
	4	6.397	$7.30\times$	182.6	Success

\* 3 out of 4 processes terminated immediately (0 iterations)

\*\* Error: "This code must be run with 4 PEs"

The MPI implementation of the Laplace code provides a unique perspective on parallel processing, particularly in how it handles process management and performance scaling. The key findings from the MPI results are:

- **MPI achieves  $7.30\times$  speedup** with 4 processes, demonstrating excellent parallel scaling
- **Super-linear efficiency** The super-linear efficiency of 182.6% suggests measurement inconsistencies between MPI and serial baselines, likely due to different optimization levels in the implementations rather than true performance gains.
- **Serial 4-process anomaly:** When serial code runs with 4 processes, 3 terminate immediately while 1 completes the work

Table 5: MPI Implementation Analysis

Metric	Value	Interpretation
Baseline Performance (1P)	46.711 s	Significantly slower than OpenMP baseline
MPI Performance (4P)	6.397 s	Strong parallel scaling
Speedup (1P $\rightarrow$ 4P)	7.30 $\times$	Excellent parallelization
Parallel Efficiency	182.6%	Super-linear efficiency <sup>†</sup>
<b>Key Ratios</b>		
Serial 4P vs Serial 1P	2.14 $\times$ faster	Unexpected improvement
MPI 4P vs Serial 4P	3.42 $\times$ faster	True MPI benefit

<sup>†</sup> Super-linear efficiency suggests different baseline implementation

The MPI implementation demonstrates strong parallel scaling with a 7.30 $\times$  speedup using 4 processes, but reveals several critical implementation characteristics. The MPI version is hard-coded to require exactly 4 processes and fails completely with any other configuration. Most notably, the MPI serial baseline (46.7s) is significantly slower than comparable OpenMP implementations, suggesting different algorithmic implementations or optimization levels. When the serial code is forced to run with 4 processes, 3 processes terminate immediately while only 1 completes the work in 21.9s, indicating the serial version lacks proper MPI coordination. The true MPI parallelization benefit is 3.42 $\times$  over the functioning 4-process baseline, which represents more realistic performance gains. While MPI achieves good parallel efficiency (85.5% based on realistic baseline), it remains less flexible than OpenMP due to its fixed process requirement and shows inferior absolute performance compared to OpenMP’s 8-thread implementation (6.4s vs 4.2s).

## Conclusion

This performance analysis reveals distinct characteristics across three computational paradigms for iterative numerical methods. **Serial execution** provides simplicity and deterministic behavior but is fundamentally limited by single-core performance (22.0-46.7s baseline). **OpenMP shared-memory parallelization** demonstrates excellent scalability with 10.91 $\times$  speedup using 32 threads, offering the best balance of performance (1.992s), programming simplicity, and flexibility, though efficiency degrades from 65.1% (8 threads) to 34.1% (32 threads) due to memory bandwidth limitations and synchronization overhead. **MPI distributed-memory parallelization** achieves strong parallel scaling (7.30 $\times$  speedup with 4 processes) and enables cluster computing, but suffers from implementation rigidity (hard-coded to exactly 4 processes), communication overhead, and reduced absolute performance (6.397s) compared to OpenMP.

The **Enhanced Red/Black checkerboard algorithm** proves that algorithmic optimization can be more impactful than parallelization paradigm choice alone. This approach delivers 25-41% per-

formance improvements across all configurations, reducing single-thread baseline from 21.7s to 12.7s and achieving superior 32-thread performance (1.490s vs 1.992s). The Red/Black method eliminates data dependencies through spatial decomposition, accelerates convergence (2.8% fewer iterations), and enables better cache utilization. Combined with OpenMP’s 32-thread parallelization, it achieves the optimal performance of 1.490s with  $14.80\times$  speedup, demonstrating that **hybrid approaches combining algorithmic innovation with appropriate parallelization strategies yield the most significant performance gains** for computationally intensive iterative methods.

For practical applications, **OpenMP with 8-16 threads provides the optimal efficiency balance (45-65%)**, while the Red/Black algorithm should be adopted universally for its consistent performance benefits. Future development should pursue **hybrid MPI+OpenMP implementations with Red/Black optimization** to combine distributed scalability with optimal single-node performance, leveraging the strengths of each paradigm while mitigating their individual limitations.

## 6 Appendix.code

Here's some of our code (Note the use of VerbatimInput from package fancyvrb):

### 6.1 Code A: laplace\_serial.c

```

/*****
 * Laplace Serial C Version
 *
 * Temperature is initially 0.0
 * Boundaries are as follows:
 *
 *      0          T          0
 * 0 +-----+ 0
 * |
 * |
 * |
 * T |          | T
 * |
 * |
 * |
 * 0 +-----+ 100
 *      0          T          100
 *
 * John Urbanic, PSC 2014
 *****/

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

// size of plate
#define COLUMNS 1000
#define ROWS 1000

// largest permitted change in temp (This value takes about 3400 steps)
#define MAX_TEMP_ERROR 0.01

double Temperature[ROWS+2][COLUMNS+2]; // temperature grid
double Temperature_last[ROWS+2][COLUMNS+2]; // temperature grid from last
iteration

// helper routines
void initialize();
void track_progress(int iter);

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



```

int i, j; // grid indexes
int max_iterations; // number of iterations
int iteration=1; // current iteration
double dt=100; // largest change in t
struct timeval start_time, stop_time, elapsed_time; // timers

printf("Maximum iterations [100-4000]? \n");
scanf("%d", &max_iterations);

gettimeofday(&start_time, NULL); // Unix timer

initialize(); // initialize Temp_last including boundary
conditions

// do until error is minimal or until max steps
while ( dt > 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]);
        }
    }

    dt = 0.0; // reset largest temperature change

    // copy grid to old grid for next iteration and find latest dt
    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];
        }
    }

    // periodically print test values
    if((iteration % 100) == 0) {
        track_progress(iteration);
    }

    iteration++;
}

gettimeofday(&stop_time, NULL);
timersub(&stop_time, &start_time, &elapsed_time); // Unix time subtract
routine

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

```

```

}

// initialize plate and boundary conditions
// Temp_last is used to to start first iteration
void initialize(){

    int i,j;

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

    // these boundary conditions never change throughout run

    // set left side to 0 and right to a linear increase
    for(i = 0; i <= ROWS+1; i++) {
        Temperature_last[i][0] = 0.0;
        Temperature_last[i][COLUMNS+1] = (100.0/ROWS)*i;
    }

    // set top to 0 and bottom to linear increase
    for(j = 0; j <= COLUMNS+1; j++) {
        Temperature_last[0][j] = 0.0;
        Temperature_last[ROWS+1][j] = (100.0/COLUMNS)*j;
    }
}

// print diagonal in bottom right corner where most action is
void track_progress(int iteration) {

    int i;

    printf("-----Iteration number: %d-----\n", iteration);
    for(i = ROWS-5; i <= ROWS; i++) {
        printf(" [%d,%d]: %5.2f\n", i, i, Temperature[i][i]);
    }
    printf("\n");
}

```

## 6.2 Code B: laplace\_omp.c

```

/*****
 * Laplace OpenMP C Version
 *
 * Temperature is initially 0.0
 * Boundaries are as follows:
 *
 */

```

```

*      0          T          0
*  0  +-----+  0
*    |         |
*    |         |
*    |         |
*  T  |         |  T
*    |         |
*    |         |
*    |         |
*  0  +-----+  100
*    0          T          100
*
*   John Urbanic, PSC 2014
*
*****/

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

// size of plate
#define COLUMNS 1000
#define ROWS 1000

// largest permitted change in temp (This value takes about 3400 steps)
#define MAX_TEMP_ERROR 0.01

double Temperature[ROWS+2][COLUMNS+2]; // temperature grid
double Temperature_last[ROWS+2][COLUMNS+2]; // temperature grid from last
iteration

// helper routines
void initialize();
void track_progress(int iter);

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

    int i, j; // grid indexes
    int max_iterations; // number of iterations
    int iteration=1; // current iteration
    double dt=100; // largest change in t
    struct timeval start_time, stop_time, elapsed_time; // timers

    printf("Maximum iterations [100-4000]? \n");
    scanf("%d", &max_iterations);

    gettimeofday(&start_time, NULL); // Unix timer

    initialize(); // initialize Temp_last including boundary
    conditions

```

```

// do until error is minimal or until max steps
while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {

    // main calculation: average my four neighbors
    #pragma omp parallel for private(i,j)
    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]);
        }
    }

    dt = 0.0; // reset largest temperature change

    // copy grid to old grid for next iteration and find latest dt
    #pragma omp parallel for reduction(max:dt) private(i,j)
    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];
        }
    }

    // periodically print test values
    if((iteration % 100) == 0) {
        track_progress(iteration);
    }

    iteration++;
}

gettimeofday(&stop_time, NULL);
timersub(&stop_time, &start_time, &elapsed_time); // Unix time subtract
routine

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

// initialize plate and boundary conditions
// Temp_last is used to to start first iteration
void initialize(){

    int i,j;

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

```

```

    }
}

// these boundary conditions never change throughout run

// set left side to 0 and right to a linear increase
for(i = 0; i <= ROWS+1; i++) {
    Temperature_last[i][0] = 0.0;
    Temperature_last[i][COLUMNS+1] = (100.0/ROWS)*i;
}

// set top to 0 and bottom to linear increase
for(j = 0; j <= COLUMNS+1; j++) {
    Temperature_last[0][j] = 0.0;
    Temperature_last[ROWS+1][j] = (100.0/COLUMNS)*j;
}

}

// print diagonal in bottom right corner where most action is
void track_progress(int iteration) {

    int i;

    printf("-----Iteration number: %d-----\n", iteration);
    for(i = ROWS-5; i <= ROWS; i++) {
        printf("[%d,%d]: %5.2f", i, i, Temperature[i][i]);
    }
    printf("\n");
}

```

### 6.3 Code C: laplace\_omp\_parallel.c

```

/*****
* Laplace OpenMP C Version
*
* Temperature is initially 0.0
* Boundaries are as follows:
*
*      0          T          0
*  0  +-----+  0
*      |         |
*      |         |
*      |         |
*  T   |         |   T
*      |         |
*      |         |
*      |         |
*  0  +-----+  100
*      0          T          100
*
*  John Urbanic, PSC 2014
*****/

```

```

*
*****/

/*****
* Optimized Laplace OpenMP C Version - Red-Black Algorithm
* Key optimizations:
* - Single grid (50% memory reduction)
* - No expensive copying/swapping
* - Better cache locality
* - SIMD-friendly inner loops
* - Optimal for modern CPUs
*****/

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

// size of plate
#define COLUMNS 1000
#define ROWS 1000

// largest permitted change in temp (This value takes about 3400 steps)
#define MAX_TEMP_ERROR 0.01

double Temperature[ROWS+2][COLUMNS+2] __attribute__((aligned(64))); //dynamic
memory allocation with 64-byte
//double Temperature_last[ROWS+2][COLUMNS+2]; // temperature grid from last
iteration

// helper routines
void initialize();
void track_progress(int iter);

#define MAX_THREADS 32

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

    int i, j; // grid indexes
    int max_iterations; // number of iterations
    int iteration=1; // current iteration
    double dt=100; // largest change in t
    struct timeval start_time, stop_time, elapsed_time; // timers

    // Set number of threads at runtime
    int num_threads = MAX_THREADS;
    if (omp_get_max_threads() < MAX_THREADS) {
        num_threads = omp_get_max_threads();
    }
    omp_set_num_threads(num_threads);
    printf("Running with %d OpenMP threads\n", num_threads);

```

```

printf("Maximum iterations [100-4000]?\n");
scanf("%d", &max_iterations);

gettimeofday(&start_time, NULL); // Unix timer
initialize(); // initialize Temp_last including boundary
               conditions

// do until error is minimal or until max steps
while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {

    dt=0.0; // reset largest temperature change

    // Process RED squares (checkerboard pattern)
    double red_dt=0.0;
    // main calculation: average my four neighbors
    #pragma omp parallel for reduction(max:red_dt) private(i,j) schedule(
        static)
    for(i = 1; i <= ROWS; i++) {
        // SIMD
        #pragma omp simd aligned(Temperature:64) reduction(max:red_dt)
        for(j = 1 + (i % 2); j <= COLUMNS; j += 2) {
            double old_temp = Temperature[i][j];

            Temperature[i][j] = 0.25 * (Temperature[i+1][j] + Temperature[i
                -1][j] +
                                         Temperature[i][j+1] + Temperature[i][j
                -1]);
            red_dt = fmax(fabs(Temperature[i][j] - old_temp), red_dt);
        }
    }

    // BLACK squares
    double black_dt = 0.0;
    // copy grid to old grid for next iteration and find latest dt
    #pragma omp parallel for reduction(max:black_dt) private(i,j) schedule(
        static)
    for(i = 1; i <= ROWS; i++){
        #pragma omp simd aligned(Temperature:64) reduction(max:black_dt)
        for(j = 1 + ((i + 1) % 2); j <= COLUMNS; j += 2){
            double old_temp = Temperature[i][j];
            Temperature[i][j] = 0.25 * (Temperature[i+1][j] + Temperature[i
                -1][j] +
                                         Temperature[i][j+1] + Temperature[i][j
                -1]);
            black_dt = fmax( fabs(Temperature[i][j]-old_temp), black_dt);
        }
    }
    dt = fmax(red_dt, black_dt);

    // periodically print test values

```

```

        if((iteration % 100) == 0) {
            track_progress(iteration);
        }

        iteration++;
    }

    gettimeofday(&stop_time, NULL);
    timersub(&stop_time, &start_time, &elapsed_time); // Unix time subtract
    routine

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

    return 0;
}

// initialize plate and boundary conditions
// Temp_last is used to to start first iteration
void initialize(){

    int i,j;

    #pragma omp parallel for private(i,j)
    for(i = 0; i <= ROWS+1; i++){
        for (j = 0; j <= COLUMNS+1; j++){
            Temperature[i][j] = 0.0;
        }
    }

    // these boundary conditions never change throughout run
    #pragma omp parallel
    {
        // set left side to 0 and right to a linear increase
        #pragma omp for
        for(i = 0; i <= ROWS+1; i++) {
            Temperature[i][0] = 0.0;
            Temperature[i][COLUMNS+1] = (100.0/ROWS)*i;
        }
        // set top to 0 and bottom to linear increase
        #pragma omp for
        for(j = 0; j <= COLUMNS+1; j++) {
            Temperature[0][j] = 0.0;
            Temperature[ROWS+1][j] = (100.0/COLUMNS)*j;
        }
    }
}

// print diagonal in bottom right corner where most action is
void track_progress(int iteration) {

    int i;

```



```

    printf("-----Iteration number: %d-----\n", iteration);
    for(i = ROWS-5; i <= ROWS; i++) {
        printf("[%d,%d]: %5.2f\n", i, i, Temperature[i][i]);
    }
    printf("\n");
}

```

## 6.4 Code D: laplace\_mpi.c

```

/*****
 * Laplace MPI C Version
 *
 * T is initially 0.0
 * Boundaries are as follows
 *
 *
 *          T                                4 sub-grids
 * 0  +-----+ 0  +-----+
 *   |         |   |         |
 *   |         |   |         |
 *   |         |   |         |
 *   |         |   |         |
 *   |         |   |         |
 *   |         |   |         |
 *   |         |   |         |
 * 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];

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 boundary 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<=10; i++){
        printf("[%d,%d]: %.2f", ROWS_GLOBAL-i, COLUMNS-i, Temperature[ROWS-i][
            COLUMNS-i]);
    }
    printf("\n");
}

```

## 7 Appendix.results

### 7.1 ex1\_result.txt

```

=== Basic System Info ===
Date: Wed Jun 18 01:02:49 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 SERIAL PROCESS TEST!!!!
=== Test with 1 threads ===
Start time: 2025-06-18 01:02:49.612249292
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 22.030055 seconds.

End time: 2025-06-18 01:03:11.656427381

Total wall clock time: 22.040 seconds

=== Test with 8 threads ===

Start time: 2025-06-18 01:03:12.666150279

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 22.026727 seconds.

End time: 2025-06-18 01:03:34.707365086

Total wall clock time: 22.037 seconds

=== Test with 32 threads ===

Start time: 2025-06-18 01:03:35.717081492

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 22.055216 seconds.

End time: 2025-06-18 01:03:57.791425777

Total wall clock time: 22.065 seconds

!!!!STARTING OMP PROCESS TEST!!!!

=== Test with 1 threads ===

Start time: 2025-06-18 01:03:58.802657573

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 21.725611 seconds.
End time: 2025-06-18 01:04:20.540366367
Total wall clock time: 21.733 seconds
-----

```

```

=== Test with 8 threads ===

```

```

Start time: 2025-06-18 01:04:21.551742656
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 4.166916 seconds.

End time: 2025-06-18 01:04:25.730579252

Total wall clock time: 4.175 seconds

=== Test with 32 threads ===

Start time: 2025-06-18 01:04:26.739732416

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 1.981397 seconds.

End time: 2025-06-18 01:04:28.735366901

Total wall clock time: 1.992 seconds

-----

!!!!STARTING Enhanced (RED/BLACK) PARALLEL PROCESS TEST!!!!

=== Test with 1 threads ===

```

Start time: 2025-06-18 01:04:29.746305624
Running with 1 OpenMP threads
Maximum iterations [100-4000]?
----- Iteration number: 100 -----
[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: 200 -----
[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: 300 -----
[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: 400 -----
[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: 500 -----
[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: 600 -----
[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: 700 -----
[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: 800 -----
[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: 900 -----
[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: 1000 -----
[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: 1100 -----
[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: 1200 -----
[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: 1300 -----
[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: 1400 -----
[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: 1500 -----
[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: 1600 -----
[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: 1700 -----
[995,995]: 97.70 [996,996]: 98.26 [997,997]: 98.76 [998,998]: 99.20 [999,999]:

```

```

    99.57 [1000,1000]: 99.87
----- Iteration number: 1800 -----
[995,995]: 97.77 [996,996]: 98.31 [997,997]: 98.80 [998,998]: 99.22 [999,999]:
    99.57 [1000,1000]: 99.87
----- Iteration number: 1900 -----
[995,995]: 97.83 [996,996]: 98.36 [997,997]: 98.82 [998,998]: 99.23 [999,999]:
    99.58 [1000,1000]: 99.87
----- Iteration number: 2000 -----
[995,995]: 97.89 [996,996]: 98.40 [997,997]: 98.85 [998,998]: 99.25 [999,999]:
    99.59 [1000,1000]: 99.87
----- Iteration number: 2100 -----
[995,995]: 97.94 [996,996]: 98.43 [997,997]: 98.87 [998,998]: 99.26 [999,999]:
    99.59 [1000,1000]: 99.87
----- Iteration number: 2200 -----
[995,995]: 97.99 [996,996]: 98.46 [997,997]: 98.89 [998,998]: 99.27 [999,999]:
    99.60 [1000,1000]: 99.87
----- Iteration number: 2300 -----
[995,995]: 98.03 [996,996]: 98.49 [997,997]: 98.91 [998,998]: 99.28 [999,999]:
    99.60 [1000,1000]: 99.88
----- Iteration number: 2400 -----
[995,995]: 98.07 [996,996]: 98.52 [997,997]: 98.93 [998,998]: 99.29 [999,999]:
    99.61 [1000,1000]: 99.88
----- Iteration number: 2500 -----
[995,995]: 98.10 [996,996]: 98.54 [997,997]: 98.94 [998,998]: 99.30 [999,999]:
    99.61 [1000,1000]: 99.88
----- Iteration number: 2600 -----
[995,995]: 98.13 [996,996]: 98.57 [997,997]: 98.96 [998,998]: 99.31 [999,999]:
    99.61 [1000,1000]: 99.88
----- Iteration number: 2700 -----
[995,995]: 98.16 [996,996]: 98.59 [997,997]: 98.97 [998,998]: 99.32 [999,999]:
    99.62 [1000,1000]: 99.88
----- Iteration number: 2800 -----
[995,995]: 98.19 [996,996]: 98.61 [997,997]: 98.98 [998,998]: 99.32 [999,999]:
    99.62 [1000,1000]: 99.88
----- Iteration number: 2900 -----
[995,995]: 98.22 [996,996]: 98.63 [997,997]: 99.00 [998,998]: 99.33 [999,999]:
    99.62 [1000,1000]: 99.88
----- Iteration number: 3000 -----
[995,995]: 98.24 [996,996]: 98.64 [997,997]: 99.01 [998,998]: 99.34 [999,999]:
    99.63 [1000,1000]: 99.88
----- Iteration number: 3100 -----
[995,995]: 98.27 [996,996]: 98.66 [997,997]: 99.02 [998,998]: 99.34 [999,999]:
    99.63 [1000,1000]: 99.88
----- Iteration number: 3200 -----
[995,995]: 98.29 [996,996]: 98.67 [997,997]: 99.03 [998,998]: 99.35 [999,999]:
    99.63 [1000,1000]: 99.88

```

```

Max error at iteration 3279 was 0.010000
Total time was 12.727960 seconds.
End time: 2025-06-18 01:04:42.489548456
Total wall clock time: 12.738 seconds
-----

```

```

=== Test with 8 threads ===
Start time: 2025-06-18 01:04:43.500290739
Running with 8 OpenMP threads
Maximum iterations [100-4000]?
----- Iteration number: 100 -----
[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: 200 -----
[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: 300 -----
[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: 400 -----
[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: 500 -----
[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: 600 -----
[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: 700 -----
[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: 800 -----
[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: 900 -----
[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: 1000 -----
[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: 1100 -----
[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: 1200 -----
[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: 1300 -----
[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: 1400 -----
[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: 1500 -----
[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: 1600 -----
[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: 1700 -----

```

```

[995,995]: 97.70 [996,996]: 98.26 [997,997]: 98.76 [998,998]: 99.20 [999,999]:
99.57 [1000,1000]: 99.87
----- Iteration number: 1800 -----
[995,995]: 97.77 [996,996]: 98.31 [997,997]: 98.80 [998,998]: 99.22 [999,999]:
99.57 [1000,1000]: 99.87
----- Iteration number: 1900 -----
[995,995]: 97.83 [996,996]: 98.36 [997,997]: 98.82 [998,998]: 99.23 [999,999]:
99.58 [1000,1000]: 99.87
----- Iteration number: 2000 -----
[995,995]: 97.89 [996,996]: 98.40 [997,997]: 98.85 [998,998]: 99.25 [999,999]:
99.59 [1000,1000]: 99.87
----- Iteration number: 2100 -----
[995,995]: 97.94 [996,996]: 98.43 [997,997]: 98.87 [998,998]: 99.26 [999,999]:
99.59 [1000,1000]: 99.87
----- Iteration number: 2200 -----
[995,995]: 97.99 [996,996]: 98.46 [997,997]: 98.89 [998,998]: 99.27 [999,999]:
99.60 [1000,1000]: 99.87
----- Iteration number: 2300 -----
[995,995]: 98.03 [996,996]: 98.49 [997,997]: 98.91 [998,998]: 99.28 [999,999]:
99.60 [1000,1000]: 99.88
----- Iteration number: 2400 -----
[995,995]: 98.07 [996,996]: 98.52 [997,997]: 98.93 [998,998]: 99.29 [999,999]:
99.61 [1000,1000]: 99.88
----- Iteration number: 2500 -----
[995,995]: 98.10 [996,996]: 98.54 [997,997]: 98.94 [998,998]: 99.30 [999,999]:
99.61 [1000,1000]: 99.88
----- Iteration number: 2600 -----
[995,995]: 98.13 [996,996]: 98.57 [997,997]: 98.96 [998,998]: 99.31 [999,999]:
99.61 [1000,1000]: 99.88
----- Iteration number: 2700 -----
[995,995]: 98.16 [996,996]: 98.59 [997,997]: 98.97 [998,998]: 99.32 [999,999]:
99.62 [1000,1000]: 99.88
----- Iteration number: 2800 -----
[995,995]: 98.19 [996,996]: 98.61 [997,997]: 98.98 [998,998]: 99.32 [999,999]:
99.62 [1000,1000]: 99.88
----- Iteration number: 2900 -----
[995,995]: 98.22 [996,996]: 98.63 [997,997]: 99.00 [998,998]: 99.33 [999,999]:
99.62 [1000,1000]: 99.88
----- Iteration number: 3000 -----
[995,995]: 98.24 [996,996]: 98.64 [997,997]: 99.01 [998,998]: 99.34 [999,999]:
99.63 [1000,1000]: 99.88
----- Iteration number: 3100 -----
[995,995]: 98.27 [996,996]: 98.66 [997,997]: 99.02 [998,998]: 99.34 [999,999]:
99.63 [1000,1000]: 99.88
----- Iteration number: 3200 -----
[995,995]: 98.29 [996,996]: 98.67 [997,997]: 99.03 [998,998]: 99.35 [999,999]:
99.63 [1000,1000]: 99.88

```

Max error at iteration 3279 was 0.010000

Total time was 2.559244 seconds.

End time: 2025-06-18 01:04:46.073213498

Total wall clock time: 2.569 seconds

-----

```

=== Test with 32 threads ===
Start time: 2025-06-18 01:04:47.083417866
Running with 32 OpenMP threads
Maximum iterations [100-4000]?
----- Iteration number: 100 -----
[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: 200 -----
[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: 300 -----
[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: 400 -----
[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: 500 -----
[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: 600 -----
[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: 700 -----
[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: 800 -----
[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: 900 -----
[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: 1000 -----
[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: 1100 -----
[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: 1200 -----
[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: 1300 -----
[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: 1400 -----
[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: 1500 -----
[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: 1600 -----
[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: 1700 -----
[995,995]: 97.70 [996,996]: 98.26 [997,997]: 98.76 [998,998]: 99.20 [999,999]:
99.57 [1000,1000]: 99.87
----- Iteration number: 1800 -----
[995,995]: 97.77 [996,996]: 98.31 [997,997]: 98.80 [998,998]: 99.22 [999,999]:
99.57 [1000,1000]: 99.87
----- Iteration number: 1900 -----
[995,995]: 97.83 [996,996]: 98.36 [997,997]: 98.82 [998,998]: 99.23 [999,999]:
99.58 [1000,1000]: 99.87
----- Iteration number: 2000 -----
[995,995]: 97.89 [996,996]: 98.40 [997,997]: 98.85 [998,998]: 99.25 [999,999]:
99.59 [1000,1000]: 99.87
----- Iteration number: 2100 -----
[995,995]: 97.94 [996,996]: 98.43 [997,997]: 98.87 [998,998]: 99.26 [999,999]:
99.59 [1000,1000]: 99.87
----- Iteration number: 2200 -----
[995,995]: 97.99 [996,996]: 98.46 [997,997]: 98.89 [998,998]: 99.27 [999,999]:
99.60 [1000,1000]: 99.87
----- Iteration number: 2300 -----
[995,995]: 98.03 [996,996]: 98.49 [997,997]: 98.91 [998,998]: 99.28 [999,999]:
99.60 [1000,1000]: 99.88
----- Iteration number: 2400 -----
[995,995]: 98.07 [996,996]: 98.52 [997,997]: 98.93 [998,998]: 99.29 [999,999]:
99.61 [1000,1000]: 99.88
----- Iteration number: 2500 -----
[995,995]: 98.10 [996,996]: 98.54 [997,997]: 98.94 [998,998]: 99.30 [999,999]:
99.61 [1000,1000]: 99.88
----- Iteration number: 2600 -----
[995,995]: 98.13 [996,996]: 98.57 [997,997]: 98.96 [998,998]: 99.31 [999,999]:
99.61 [1000,1000]: 99.88
----- Iteration number: 2700 -----
[995,995]: 98.16 [996,996]: 98.59 [997,997]: 98.97 [998,998]: 99.32 [999,999]:
99.62 [1000,1000]: 99.88
----- Iteration number: 2800 -----
[995,995]: 98.19 [996,996]: 98.61 [997,997]: 98.98 [998,998]: 99.32 [999,999]:
99.62 [1000,1000]: 99.88
----- Iteration number: 2900 -----
[995,995]: 98.22 [996,996]: 98.63 [997,997]: 99.00 [998,998]: 99.33 [999,999]:
99.62 [1000,1000]: 99.88
----- Iteration number: 3000 -----
[995,995]: 98.24 [996,996]: 98.64 [997,997]: 99.01 [998,998]: 99.34 [999,999]:
99.63 [1000,1000]: 99.88
----- Iteration number: 3100 -----
[995,995]: 98.27 [996,996]: 98.66 [997,997]: 99.02 [998,998]: 99.34 [999,999]:
99.63 [1000,1000]: 99.88
----- Iteration number: 3200 -----
[995,995]: 98.29 [996,996]: 98.67 [997,997]: 99.03 [998,998]: 99.35 [999,999]:
99.63 [1000,1000]: 99.88

```

Max error at iteration 3279 was 0.010000  
 Total time was 1.480047 seconds.  
 End time: 2025-06-18 01:04:48.578160632  
 Total wall clock time: 1.490 seconds

## 7.2 ex2\_result.txt

```
-----

=== Basic System Info ===
Date: Wed Jun 25 14:19:21 CDT 2025
System: Linux cn034.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 SERIAL process TEST!!!!
=== Test with 1 processes ===
Start time: 2025-06-25 14:19:21.576658302
[cn034.delta.ncsa.illinois.edu:17094] mca: base: components_register: registering
      framework ras components
[cn034.delta.ncsa.illinois.edu:17094] mca: base: components_register: found loaded
      component simulator
[cn034.delta.ncsa.illinois.edu:17094] mca: base: components_register: component
      simulator register function successful
[cn034.delta.ncsa.illinois.edu:17094] mca: base: components_register: found loaded
      component slurm
[cn034.delta.ncsa.illinois.edu:17094] mca: base: components_register: component
      slurm register function successful
[cn034.delta.ncsa.illinois.edu:17094] mca: base: components_open: opening ras
      components
[cn034.delta.ncsa.illinois.edu:17094] mca: base: components_open: found loaded
      component simulator
[cn034.delta.ncsa.illinois.edu:17094] mca: base: components_open: found loaded
      component slurm
[cn034.delta.ncsa.illinois.edu:17094] mca: base: components_open: component slurm
      open function successful
[cn034.delta.ncsa.illinois.edu:17094] mca:base:select: Auto-selecting ras
      components
[cn034.delta.ncsa.illinois.edu:17094] mca:base:select:(  ras) Querying component [
      simulator]
[cn034.delta.ncsa.illinois.edu:17094] mca:base:select:(  ras) Querying component [
      slurm]
[cn034.delta.ncsa.illinois.edu:17094] mca:base:select:(  ras) Query of component [
      slurm] set priority to 50
[cn034.delta.ncsa.illinois.edu:17094] mca:base:select:(  ras) Selected component [
      slurm]
[cn034.delta.ncsa.illinois.edu:17094] mca: base: close: unloading component
      simulator
[cn034.delta.ncsa.illinois.edu:17094] mca: base: close: component slurm closed
[cn034.delta.ncsa.illinois.edu:17094] mca: base: close: unloading component slurm
End time: 2025-06-25 14:20:08.292080598
Total wall clock time:
===== ALLOCATED NODES =====
      cn034: flags=0x11 slots=4 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 46.471284 seconds.
46.711 seconds
-----

```

```

=== Test with 4 processes ===
Start time: 2025-06-25 14:20:09.303380827

```

```

[cn034.delta.ncsa.illinois.edu:17423] mca: base: components_register: registering
framework ras components
[cn034.delta.ncsa.illinois.edu:17423] mca: base: components_register: found loaded
component simulator
[cn034.delta.ncsa.illinois.edu:17423] mca: base: components_register: component
simulator register function successful
[cn034.delta.ncsa.illinois.edu:17423] mca: base: components_register: found loaded
component slurm
[cn034.delta.ncsa.illinois.edu:17423] mca: base: components_register: component
slurm register function successful
[cn034.delta.ncsa.illinois.edu:17423] mca: base: components_open: opening ras
components
[cn034.delta.ncsa.illinois.edu:17423] mca: base: components_open: found loaded
component simulator
[cn034.delta.ncsa.illinois.edu:17423] mca: base: components_open: found loaded
component slurm
[cn034.delta.ncsa.illinois.edu:17423] mca: base: components_open: component slurm
open function successful
[cn034.delta.ncsa.illinois.edu:17423] mca:base:select: Auto-selecting ras
components
[cn034.delta.ncsa.illinois.edu:17423] mca:base:select:( ras) Querying component [
simulator]
[cn034.delta.ncsa.illinois.edu:17423] mca:base:select:( ras) Querying component [
slurm]
[cn034.delta.ncsa.illinois.edu:17423] mca:base:select:( ras) Query of component [
slurm] set priority to 50
[cn034.delta.ncsa.illinois.edu:17423] mca:base:select:( ras) Selected component [
slurm]
[cn034.delta.ncsa.illinois.edu:17423] mca: base: close: unloading component
simulator
[cn034.delta.ncsa.illinois.edu:17423] mca: base: close: component slurm closed
[cn034.delta.ncsa.illinois.edu:17423] mca: base: close: unloading component slurm
End time: 2025-06-25 14:20:31.195120773
Total wall clock time:
===== ALLOCATED NODES =====
cn034: flags=0x11 slots=4 max_slots=0 slots_inuse=0 state=UP
=====
Maximum iterations [100-4000]?
Maximum iterations [100-4000]?
Maximum iterations [100-4000]?

Max error at iteration 0 was 100.000000
Total time was 0.006821 seconds.

Max error at iteration 0 was 100.000000
Total time was 0.007908 seconds.
Maximum iterations [100-4000]?

Max error at iteration 0 was 100.000000
Total time was 0.007612 seconds.
----- 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 21.628182 seconds.

21.878 seconds

!!!!STARTING MPI TEST!!!!

=== Test with 1 processes ===

Start time: 2025-06-25 14:20:32.208041495

[cn034.delta.ncsa.illinois.edu:17436] mca: base: components\_register: registering

```

framework ras components
[cn034.delta.ncsa.illinois.edu:17436] mca: base: components_register: found loaded
component simulator
[cn034.delta.ncsa.illinois.edu:17436] mca: base: components_register: component
simulator register function successful
[cn034.delta.ncsa.illinois.edu:17436] mca: base: components_register: found loaded
component slurm
[cn034.delta.ncsa.illinois.edu:17436] mca: base: components_register: component
slurm register function successful
[cn034.delta.ncsa.illinois.edu:17436] mca: base: components_open: opening ras
components
[cn034.delta.ncsa.illinois.edu:17436] mca: base: components_open: found loaded
component simulator
[cn034.delta.ncsa.illinois.edu:17436] mca: base: components_open: found loaded
component slurm
[cn034.delta.ncsa.illinois.edu:17436] mca: base: components_open: component slurm
open function successful
[cn034.delta.ncsa.illinois.edu:17436] mca:base:select: Auto-selecting ras
components
[cn034.delta.ncsa.illinois.edu:17436] mca:base:select:( ras) Querying component [
simulator]
[cn034.delta.ncsa.illinois.edu:17436] mca:base:select:( ras) Querying component [
slurm]
[cn034.delta.ncsa.illinois.edu:17436] mca:base:select:( ras) Query of component [
slurm] set priority to 50
[cn034.delta.ncsa.illinois.edu:17436] mca:base:select:( ras) Selected component [
slurm]
[cn034.delta.ncsa.illinois.edu:17436] mca: base: close: unloading component
simulator
-----
Primary job terminated normally, but 1 process returned
a non-zero exit code. Per user-direction, the job has been aborted.
-----
mpirun detected that one or more processes exited with non-zero status, thus
causing
the job to be terminated. The first process to do so was:

Process name: [[47462,1],0]
Exit code: 1
-----
[cn034.delta.ncsa.illinois.edu:17436] mca: base: close: component slurm closed
[cn034.delta.ncsa.illinois.edu:17436] mca: base: close: unloading component slurm
End time: 2025-06-25 14:20:34.590819048
Total wall clock time:
===== ALLOCATED NODES =====
cn034: flags=0x11 slots=4 max_slots=0 slots_inuse=0 state=UP
=====
This code must be run with 4 PEs
2.378 seconds
-----

=== Test with 4 processes ===

```

```

Start time: 2025-06-25 14:20:35.602046300
[cn034.delta.ncsa.illinois.edu:17448] mca: base: components_register: registering
framework ras components
[cn034.delta.ncsa.illinois.edu:17448] mca: base: components_register: found loaded
component simulator
[cn034.delta.ncsa.illinois.edu:17448] mca: base: components_register: component
simulator register function successful
[cn034.delta.ncsa.illinois.edu:17448] mca: base: components_register: found loaded
component slurm
[cn034.delta.ncsa.illinois.edu:17448] mca: base: components_register: component
slurm register function successful
[cn034.delta.ncsa.illinois.edu:17448] mca: base: components_open: opening ras
components
[cn034.delta.ncsa.illinois.edu:17448] mca: base: components_open: found loaded
component simulator
[cn034.delta.ncsa.illinois.edu:17448] mca: base: components_open: found loaded
component slurm
[cn034.delta.ncsa.illinois.edu:17448] mca: base: components_open: component slurm
open function successful
[cn034.delta.ncsa.illinois.edu:17448] mca:base:select: Auto-selecting ras
components
[cn034.delta.ncsa.illinois.edu:17448] mca:base:select:( ras) Querying component [
simulator]
[cn034.delta.ncsa.illinois.edu:17448] mca:base:select:( ras) Querying component [
slurm]
[cn034.delta.ncsa.illinois.edu:17448] mca:base:select:( ras) Query of component [
slurm] set priority to 50
[cn034.delta.ncsa.illinois.edu:17448] mca:base:select:( ras) Selected component [
slurm]
[cn034.delta.ncsa.illinois.edu:17448] mca: base: close: unloading component
simulator
[cn034.delta.ncsa.illinois.edu:17448] mca: base: close: component slurm closed
[cn034.delta.ncsa.illinois.edu:17448] mca: base: close: unloading component slurm
End time: 2025-06-25 14:20:42.004182996
Total wall clock time:
===== ALLOCATED NODES =====
cn034: flags=0x11 slots=4 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 6.016411 seconds.  
 6.397 seconds

-----