Exercise Sheet 7 for Lecture Parallel Programming

Deadline: 2025-06-30 and 2025-07-07, 23:59

Prof. Dr. Michael Kuhn (michael.kuhn@ovgu.de)
Michael Blesel (michael.blesel@ovgu.de)
Parallel Computing and I/O • Institute for Intelligent Cooperating Systems
Faculty of Computer Science • Otto von Guericke University Magdeburg
https://parcio.ovgu.de

---

1. Parallelization with MPI (Deadline 2) (180 Points)

Parallelize the Jacobi method of the `partdiff` program with MPI. The results shall remain the same when compared to the serial version.

There are two cases to think about:

1. Jacobi method with termination after iterations
2. Jacobi method with termination after precision

Make sure that the results with multiple processes are identical to the serial program version. If that is not the case, your program is wrong!

Requirements and Hints

- The program must not be slower than the serial version.
- At **no point** is one process allowed to store the complete matrix in its memory.
- The program still has to work with one process.
- The program must work with an arbitrary amount of processes.
- Create a separate function for the MPI-parallelized Jacobi method.

  Hint: Use the `displayMatrixMpi` function that is provided in the materials for printing out the distributed matrix. Also take a look at its comments to understand how the memory layout for each process is supposed to look.

Hint: It is hard to gauge the time that this task will take, because it heavily relies on your previous knowledge and also luck that your first try will yield a working MPI implementation. Debugging complex errors for this task can take a lot of time, therefore **start early**.

1.1. Data Distribution (Deadline 1) (120 Points)

The first step in getting a correct MPI parallelization for the `partdiff` application is getting the distribution of the matrix data for the individual MPI processes correct. Take a closer look at the comments for the provided `displayMatrixMpi` function to get an understanding of what data of the global matrix every MPI process should allocate and initialize.

Implement the `allocateMatrices` and `initMatrices` functions according to the data distribution requirements stated in Task 1. Afterwards make sure that the provided `displayMatrixMpi` function works correctly with your code and add a call to this function to the `main` function of the program. For this task you can remove the call to the `calculate` function from your program. You are only supposed to show that your MPI processes are allocating the correct amount of memory and that your code is able to display the matrix correctly.

Hint: since the `initMatrices` function initializes the whole matrix with zeroes you might want to modify it a bit for testing purposes to make it clearer for yourself if in fact the displayed output shows the correct lines from the global matrix.

2. Performance Analysis (Deadline 2) (120 Points)

Measure the performance of your program and visualize the runtime for the following configurations in an appropriate diagram. A configuration (N, P, I) in the following means that your programm shall run on N nodes with a total of P evenly distributed processes and I interlines.

(1, 1, 836), (1, 2, 1,182), (1, 3, 1,448), (1, 6, 2,048), (1, 12, 2,896), (1, 24, 4,096),
(2, 48, 5,793), (4, 96, 8,192), (8, 192, 11,585)

Also compare your program to the original serial version of the program. The shortest run should take at least 30 seconds; choose appropriate parameters!

Repeat each measurement at least three times to get sensible averages. For this, you can use the `hyperfine` tool which can be loaded with `module load hyperfine`.

For `N` nodes, `P` processes, `n` iterations and `I` iterations you can use the following script:

```
1 #!/bin/bash
2
3 #SBATCH --nodes=N
4 #SBATCH --ntasks=P
5 #SBATCH --exclusive
6 #SBATCH --partition=vl-parcio
7
8 srun --mpi=pmi2 ./partdiff 1 2 I 2 2 n
```

The measurements should be done with SLURM on the compute nodes of the cluster. Document the hardware specifications used for the measurements (processor, core count, available main memory, etc.).

In the materials, you can find prepared job scripts.

Visualize all results in appropriately labeled diagrams. Write about a quarter page of interpetation for these results.

3. Hybrid Parallelization (Deadline 2) (120 Bonus Points)

Also parallelize your MPI version of the Jacobi method with OpenMP.

Measure the performance of your hybrid program and compare the runtimes of the following configurations in a diagram:

- 1 process × 24 threads
- 2 processes × 12 threads
- 3 processes × 8 threads
- 6 processes × 4 threads
- 12 processes × 2 threads
- 24 processes × 1 thread

Use 4,096 interlines for this. The shortest run should take at least 30 seconds; choose appropriate parameters!

Repeat each measurement at least three times to get sensible averages.

The measurements should be done with SLURM on one of the compute nodes of the cluster. Document the hardware specifications used for the measurements (processor, core count, available main memory, etc.) and use the same compute node for each measurement.

Visualize all results in appropriately labeled diagrams. Write about a quarter page of interpetation for these results.

4. Gauß-Seidel Parallelization (Deadline 2) (300 Bonus Points)

If you are confident in your Jacobi parallelization and want to challenge yourself, try to also parallelize the Gauß-Seidel method.

For termination after iterations, the program shall still yield the exact same results as the serial version. For termination after precision, the program does not have to stop after the exact same number of iterations as the serial version. However, the results must still be the correct ones for the number of iterations that have been performed.

Create a separate function for the Gauß-Seidel method and make sure that this task does not break your Jacobi implementation. The same requirements as for the Jacobi version apply.

Submission

We will count your last commit on the `main` branch of your repository before the exercise deadline as your submission. In the root directory of the repository, we expect a `PP-2025-Exercise-07-Materials` directory with the following contents:

- A file `group.md` with your group members (one per line) in the following format:

  ```
  Erika Musterfrau <erika.musterfrau@example.com>

  Max Mustermann <max.mustermann@example.com>
  ```

- **Deadline 1**:
  - The modified Code of the `partdiff` program in the `pde` directory (Task 1.1)
- **Deadline 2**:
  - The modified Code of the `partdiff` program in the `pde` directory (Tasks 1 , 3 and 4)
    - Optional: A `partdiff-hybrid` target in the Makefile which creates a `partdiff-hybrid` binary with the hybrid parallelization
  - A `performance-analysis.pdf` document with the measured runtimes and your interpretations (Tasks 2 and 3)