Examination 2—MPI

Regulations

Groups

The tasks must be completed together with your team partner. However, the submission must clearly indicate which student authored which parts of the solution. It is sufficient to include a short comment inside the source code, and marking sections as "created together" is also appropriate. Moreover, each team partner must be able to understand and explain **all** parts of the submitted solution.

Solutions and ideas must **not** be exchanged with other groups. If identical or nearly identical solutions are submitted by different groups, all affected submissions may be marked as failed. Submissions will also be compared against solutions from previous semesters.

Questions

If you have content-related or organizational questions, please post them in the discussion forum on the e-learning platform. Make sure not to reveal any details of your own solution. If posting publicly is not possible in exceptional cases, please send an email to jonas.posner@cs.hs-fulda.de.

Cluster

All programs must run correctly on the HPC cluster of Fulda University of Applied Sciences. For all measurements, the compiler `gcc/14.3.0` must be used.

Reproducibility

For each program, you must provide the complete command lines for compilation and execution (including any required environment variables), as well as all Slurm scripts used to run your jobs. Please name all files exactly as specified in the assignment. In addition, submit all Slurm output files containing the measured times used in your speedup tables. Non-reproducible solutions may be marked as failed.

Submission format

Please submit a single `.zip` file via the e-learning platform. The archive must contain:

- all `.c` files (your implementations),
- the `Makefile`,
- all Slurm scripts used for running the programs,
- all Slurm output files containing the measured execution times,
- a `README.md` with instructions for compilation and execution,
- a PDF document containing your speedup plots and tables.

Submissions that do not follow this structure may be marked as failed.

Submission deadline

The solutions must be submitted no later than 20th February 2026, 09:00 AMvia the e-learning platform. Submissions received after this deadline will be treated as not submitted.

Usage of third-party sources

All external sources used (e.g., websites, books, articles, presentation slides, etc.) must be cited. Solutions created fully or partially with the help of AI tools (e.g., ChatGPT) must also be referenced accordingly. Failure to properly cite sources will be considered plagiarism and may be regarded as an attempt to deceive by the entire group. This applies in particular to code.

Please submit your solutions by **20th February 2026, 09:00 AM**!

Task 2.1 (`matmul.c`)

Each team member must write **one *efficient*** parallel MPI program that performs parallel matrix multiplication. This means: teams consisting of one member submit one program; teams consisting of two members submit two programs. Each program computes the product $C = A \times B$ of two square matrices A and B of size $n \times n$. If a team has two members, the two programs must use **different parallelization strategies**.

Only MPI is allowed to be used for parallelization (i.e., OpenMP or other parallelization approaches are **not** allowed). Take into account **all** efficiency aspects discussed in the lecture. Design and implement parallelization strategies that achieve good performance and scalability across multiple nodes.

As a starting point, you may use the implementation `row_wise_matrix_mult.c` from Moodle. However, it has **very poor performance** and is intended only for comparing and verifying the computed results.

Matrix Initialization

To ensure comparability of results, the matrices A and B must be initialized using the following function (your C code may differ as long as the resulting values are identical):

```
void fillArray(double** arr, int value){
    for(int i=0; i<MATRIX_SIZE; i++){
        for(int j = 0; j<MATRIX_SIZE; j++){
            unsigned long state = concatenate(i, j) + SEED + value;
            arr[i][j] = my_rand(&state, 0, 1);
        }
    }
}
```

For matrix A, use `value = 0`; for matrix B, use `value = 1`. The implementations of `my_rand` and `concatenate` are provided in `row_wise_matrix_mult.c` on Moodle.

Verification

To verify correctness, the program must compute and output a checksum of the result matrix:

$$\text{checksum} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} C[i][j] \mod 2^{64}$$

Program Output

At the end, the program must output the following information:

- the matrices A, B, and C (if `verbose=1` and $n \leq 10$),
- the checksum of matrix C (always),
- the total execution time (always).

Note: At the end of the program, rank 0 must contain the complete matrix C.
The output must be formatted **exactly** as shown in the examples below. The timing measurement must start immediately after reading the command-line parameters (i.e., immediately after `MPI_Init`) and end immediately before `MPI_Finalize`. Both the matrix initialization and all output must occur within the timed region.

Program Invocation

```
mpirun -np <num_processes> ./matmul n seed verbose
```

`n`	Size of the square matrices ($n \times n$)
`seed`	Initial value for the random number generator
`verbose`	If set to `1` and $n \leq 10$, the matrices A, B, and C are printed. If set to `0`, only the checksum is printed. You are free to add additional verbosity levels.

Example 1:

```
mpirun -np 4 ./matmul 4 42 1

Matrix A:
0.339085 0.619920 0.628749 0.618386
0.331416 0.321054 0.893086 0.882724
0.518990 0.508628 0.303022 0.583857
0.840900 0.121735 0.820175 0.101010

Matrix B:
0.619920 0.628749 0.618386 0.608024
0.321054 0.893086 0.882724 0.435565
0.508628 0.303022 0.583857 0.864692
0.121735 0.820175 0.101010 0.109839

Matrix C (Result):
0.804312 1.464552 1.186467 1.087785
0.870234 1.489719 1.098944 1.210551
0.710231 1.351251 1.005812 0.863250
0.989835 0.968812 1.116528 1.284605
```

```
Checksum: 17.502887
Execution time with 4 ranks: 0.50 s
```

Example 2:

```
mpirun -np 4 ./matmul 4 42 0

Checksum: 17.502887
Execution time with 4 ranks: 0.50 s
```

Note: The execution times shown are approximate and may vary. The checksums must match exactly.

Speedup Measurement

Determine the speedup of your program(s) using the following configuration:

- Matrix size: $n = 8000$
- Seed: 42
- Verbose: 0

Measure the execution time for the following setups on the Fulda HPC Cluster:

Number of Nodes	MPI Processes per Node
1	64
2	64
4	64
6	64
8	64

Submit the measured execution times and speedup values, including a graphical plot. Use the single-node configuration (1 node, 64 processes) as the baseline for speedup calculation. As discussed in the lecture, the execution times used for speedup calculation must be **averaged over multiple runs**.

Please do not submit any programs to run on the Fulda HPC Cluster with a matrix size greater than 8000!

Performance Analysis

In addition to the speedup measurements from the previous section, perform a performance analysis of your implementation(s). You are free to decide how to conduct this analysis; there are no specific guidelines.

You may use tools introduced in the lecture to analyze performance, such as EduMPI and/or Score-P/Scalasca in conjunction with Cube and TAU. Alternatively or additionally, you may perform your own runtime measurements with different parameters for your program or parts of it, and create tables and graphs. You may also create variants of your program and compare them.

1. First, take a closer look at `row_wise_matrix_mult.c` and analyze it. This step is mandatory.

2. Second, analyze your own implementation(s). Compare your implementation(s) with `row_wise_matrix_mult.c` (and with each other, if applicable), and draw a conclusion as to which implementation performs best under which conditions. Your conclusions must be *justified based on your performance analysis results*. Unsubstantiated or generic statements that do not reference your measured data are insufficient. Discuss how the implementations differ in terms of scalability, memory efficiency, and MPI communication.

Submit a PDF document containing this analysis, with 0.5–1 pages of text *plus* any supporting figures or screenshots. You must use the following minimal LaTeX template:

```
\documentclass[a4paper,11pt]{article}
\usepackage[utf8]{inputenc}
\usepackage[margin=2.5cm]{geometry}

\title{Performance Analysis -- Parallel Matrix Multiplication}
\author{Your Name(s)}
\date{\today}

\begin{document}
\maketitle

\section*{Analysis of row_wise_matrix_mult.c}
% Your analysis here...

\section*{Analysis of Your Implementation(s)}
% Your analysis and comparison here...

\end{document}
```