Winter Term
2024/2025

Prof. Dr. Christoph Pflaum (Lecture)
Dominik Thönnes
Marcus Fehr (Board Exercise)
Sebastian Bönning (Computer Exercise)

Simulation and Scientific Computing
Assignment 1

Organizational Details

1. **Your tutor:** Your tutor for the board exercises is Marcus Fehr. You can find more information (office room number, e-mail address) on the official web page of the Chair for System Simulation:

   `https://www.cs10.tf.fau.de`

   Your tutor for the computer exercises is Sebastian Bönning.

2. **Programming exercises:** The programming language for the programming exercises is C++.

3. **Team work:** You have to form teams of three submitting your solution together. It is each student's responsibility to form a team of three. Before you submit your solution, be sure to have formed your team on StudOn, too.

4. **Credits:** In each of the four assignment you can achieve a maximum of 5 points (plus a possible bonus point). In order to pass the exercise classes, you have to acquire 13 or more points out of 20.

5. **Seminar:** The SiWiR seminar ("tutorium") consists of several presentation sessions. Each participant is required to give a presentation and to attend every session in order to pass in the tutorium. It is organized by Prof. Pflaum and thus questions, etc. should be addressed directly to him.

6. **Plagiarism:** Each team must submit their original solution. If a team ignores this, their submission will be graded with zero points.

7. **Computer exercises:** We offer computer exercises where you can ask questions and get help when you struggle with parts of the assignment. Make use of this opportunity!

8. **StudOn forum:** There is a forum available in the StudOn course that you can use to find team members, discuss the assignment with other students, etc.

9. **Reference system:** The nodes of the Meggie cluster act as a reference system (performance, compilation, etc.). Access is granted for all StudOn course members.

Assignment 1

1. Implement a matrix-matrix multiplication $C = A \cdot B$, where A is a $M \times K$ matrix, B is a $K \times N$ matrix, and C is a $M \times N$ matrix.

   Use any algorithm and programming technique from the lecture to decrease the single-core runtime of your program and adhere to the following guidelines:

   - All three matrices are represented as a linearized one-dimensional array with adjacent elements.
   - All three matrices are passed to your multiplication routine in a row-major format. Meaning, it is not allowed to store one of the input matrices in a transposed layout. However, the computation of the transpose is allowed to be a part of the multiplication routine itself such that the matrix is transposed after the start of the time measurement.
   - Use double precision floating-point operations for your multiplication.
   - Threads may *not* be used.
   - Use exactly two optimization techniques from the lecture or exercise.

   Compare your results to the provided reference matrix files on StudOn together with the corresponding input files! Make sure your implementation also works with non-square matrices! You can transfer data from your local machine to your Meggie home directory using scp:

   ```
   scp -r ./path/to/file/or/folder
   <hpc-account>@meggie.rrze.fau.de:~
   ```

   Use the provided Timer class (`Timer.h`) to measure the computation time.

2. Use *likwid* (Like I knew what I am doing[1]) lightweight performance tools to measure the performance of your program and to gather information about the following events:

   - L2 bandwidth,
   - L2 miss rate,
   - double precision FLOPS.

   A documentation for *likwid* can be found on the project's web page. A short introduction for the necessary *likwid* commands will be given in the first exercise class.

   For your performance and *likwid* measurements, you have to use one of the nodes of the Meggie cluster. After activation by our administrator, you can access the front end of Meggie via ssh:

   ```
   ssh <hpc-account>@meggie.rrze.fau.de
   ```

   Important note: Make sure that only your program is currently executing in order to measure reliable data! This can be done by allocating your own node.

   You can start an interactive session on one compute node with the command:

   ```
   salloc --nodes=1 --time=00:30:00 --constraint=hwperf
   ```

[1] `https://github.com/RRZE-HPC/likwid`

Due to limited resources, it is only allowed to use as many nodes as needed (one in this assignment).
Most of the development can be done on the front end of the Meggie cluster. You do not have to start an interactive session for that. You should also compile your code on the front end. Once your program is finished and compiled, you can start an interactive session and do your measurements. On the StudOn page, an example code, Makefile and likwid script can be found that show you how to compile and run a program that uses the *likwid* tool's marker API on the Meggie cluster. Likwid allows you to measure named regions in the code.

3. Provide measurements and corresponding graphs that show the effect of the used optimization techniques on runtime, double precision FLOPS, L2 bandwidth and cache misses for the matrix-matrix multiplication. Perform the measurements for matrix sizes of 32^2, 64^2, 128^2, 256^2, 512^2, 1024^2, and 2048^2. Plot well-labeled diagrams with the matrix size in $\log_2$ scale on the abscissa: one for L2 bandwidth, one for the L2 cache miss rate, one for the double precision FLOPS, and one for runtime. In each diagram, display graphs for the standard implementation (without optimization), each single optimization, both optimizations combined, and the MKL implementation of the BLAS library function `dgemm`. The code example on StudOn also shows how to use and compile the BLAS library and how to measure the time using Timer.h. Please also use the command *mkl_set_num_threads(1)* as shown in the example. Print all these diagrams in a PDF file, together with a description of information you can derive from the graphs (e.g. salient points) and with a comparison of the different implementations. You are encouraged to use information gained while optimizing the code with the help of the *likwid* tool.

4. Your program must be callable in the following form:

```
./matmult A.in B.in C.out VAR
```

where A.in and B.in are two files containing the two matrices which are multiplied and C.out is the output file for the resulting matrix. Use the following file format: the first line contains the number of rows and the number of columns of the matrix. From the second line on, each line contains exactly one element in the row by row order $x_{11}, x_{12}, \ldots, x_{1n}$, $x_{21}, \ldots, x_{2n}$, $x_{31}, \ldots, x_{mn}$. The beginning of a file for a 10×20 matrix might for example look like this:

10 20
0.1892
0.2783
0.4657
...

The last argument `VAR` is a string that denotes the algorithm or optimization variant that the code should apply. Specifying `STD` should use the standard, unoptimized matrix-matrix multiplication. Specifying `BLAS` should use the BLAS library function. You should then define two other options, like e.g. `OPT1` etc., to allow switching between your two implemented optimizations. **Document the available options and the underlying optimizations in a README file.** If the `VAR` argument is not provided, the program should be executed by default with your fastest optimization (not the BLAS library function).

Submission requirements

Hand in your solution on StudOn before the deadline. Up to 5 points are awarded for a successful submission. For this, make sure the following requirements are met:

- The program must compile with a Makefile or CMakeLists.txt you provide. Your program must compile successfully (i.e. neither errors nor warnings!) when calling `make` on the **front end** of the Meggie cluster with (at least) the following g++ compiler flags:

  ```
  -std=c++17 -Wall -Wextra -Wshadow -Werror -O3 -DNDEBUG
  ```

  Additional compiler flags are optional.

  The reference compiler is GCC version 8.5.0 on the Meggie cluster. You can check the version by typing `g++ --version`.

- The program must be callable as specified in 4.

- The program must be compiled with *likwid* support by default and must output the elapsed wall clock time of the actual computation of the multiplication. Measure the time of your matrix-matrix multiplication, but not the time for loading the matrices from disk and storing the result to disk.

- Your submission must contain well-commented source files, a PDF file with performance graphs and instructions how to use your program via a README file. Upload your solution to StudOn as a team submission.

Don't upload the `matrices.tar.bz2` archive or the extracted version of it!

Performance challenge

Your code is tested and the performance is measured. The program is run several times on one of the nodes of the Meggie cluster with a problem size of 2048^2. The team with shortest runtime, and of course the correct result, is awarded with one extra point.