Prof. Dr. Christoph Pflaum
Christian Kuschel
Christoph Rettinger

Winter Term
2016/2017

Simulation and Scientific Computing
Assignment 1

Organizational Details

1. **Your Tutors:** Your tutors for the exercises are Christian Kuschel and Christoph Rettinger. You can find their offices and e-mail addresses on the official web page of the chair for system simulation:

   https://www10.cs.fau.de

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

3. **Assignment sheets:** The assignment sheets and further information concerning this lecture are available on the web page for this lecture:

   https://www.studon.fau.de/studon/goto.php?target=crs_1661464

4. **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.

5. **Credits:** Each assignment will be graded with five points maximum. In order to pass the exercise classes, you have to acquire 13 or more points out of 20.

6. **Tutorial:** The SiWiR tutorial consists of several presentation sessions. Each participant is required to give a presentation and to attend every session in order to pass in the tutorial.

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

Exercise 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 at least two optimization techniques from the lecture or exercises.

   Compare your results to the provided reference files on StudOn together with the corresponding input files! Make sure your implementation also works with non-square matrices!

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

2. Use *likwid* (Like I knew what I am doing – http://code.google.com/p/likwid) 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.

   Important note: Make sure that only your program is currently executing in order to measure reliable data!

   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 exercise classes.

   For your performance and *likwid* measurements, you have to use one of the computers in the LSS CIP pool. On the StudOn page, an example Makefile can be found that shows you how to compile your program that uses the *likwid* tool's marker API, which 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, and 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, all optimizations combined, and the ATLAS implementation of the BLAS library function `dgemm`. 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`,

   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
   ...

Hand in your solution by Friday, November 11, 2016, 23:59. Make sure the following requirements are met:

- The program must compile with a Makefile you provide. Your program must compile successfully (i.e. neither errors nor warnings!) when calling `make` on the LSS CIP pool with the following g++ compiler flags:

  `-std=c++11 -Wall -Wextra -Wshadow -Werror -O3 -DNDEBUG`

  Additional compiler flags are optional.

  The reference compiler is GCC version 5.4.0 on the CIP Pool computers. 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 code must be well-commented source files, a PDF file with performance graphs and, if necessary, instructions how to use your program (e.g. a README file). Upload your solution to StudOn as a team submission.

Performance Challenge

Your code is tested and the performance is measured. Every code is compiled with the g++ compiler version 5.4.0. The program is run several times on one of the LSS CIP pool computers with a problem size of 2048^2. The team with shortest runtime is awarded with an extra credit point.

Beat Your Tutors!

The ultimate performance challenge: if you win the performance challenge and also beat the time of `/software/siwir/matmult`, you may choose to present your implementation in the tutorial.