



Prof. Dr. Christoph Pflaum
Florian Schornbaum
Christian Kuschel

Winter Term
2015/2016

Simulation and Scientific Computing Assignment 1

Exercise 1 (*Matrix-Matrix Multiplication*)

Organizational Details

1. **Your Tutors:** Your tutors for the exercises are Florian Schornbaum and Christian Kuschel. 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 implementation of 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:

http://www.studon.uni-erlangen.de/studon/goto.php?target=crs_1335364

4. **Team work:** You have to form teams of three submitting their solution together. The team formation is carried out in individual responsibility of the students. In front of the computer exercise room 00.131-128 at the chair for System Simulation a corresponding list is available for sign up. Each team member must sign up *by October 31, 2015, at latest*. Before you submit your solution, be sure to have formed your team on StudOn, too.
5. **Credits:** Each assignment will be graded with two points maximum. In order to pass in the exercise classes, you have to acquire five or more points.
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 successfully 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.
8. **Remote Access:** Remote access to our computers is only possible via the *secure shell (ssh)*. To login you first have to either access a computer from the university network, or log in via VPN, from where you can access the LSS CIP pool machines *i10cip{1 ... 12}*. In our computer exercise room, the same rules as in the CIP pool of the computer science department apply:

<https://wwwcip.informatik.uni-erlangen.de/cipPools/rules.en.html>.

Tasks

1. Your task is to 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.

Feel free to use any known algorithm and programming technique to decrease the single-core runtime of your program as long as you 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.
 - Make sure you use double precision floating-point operations for your multiplication.
 - The use of threads is prohibited.
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 of this lecture, 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. Your program should be able to be executed 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 $x_{11}, x_{12}, \dots, x_{1n}, x_{21}, \dots, x_{2n}, x_{31}, \dots, x_{mn}$. The beginning of a file for a 10×20 matrix might for example look like this:

```
10 20
0.1892
0.2783
0.4657
...
```

4. You must use at least two optimization techniques from the lecture or exercises.
5. Compare your calculation results to the reference files provided on the course website on StudOn together with the corresponding input files! Make sure your implementation also works with non-square matrices.
6. Please hand in your solution to this exercise until Monday, November 9, 2015, 08:00 am. Make sure the following requirements are met:
 - (a) The program must compile with a Makefile. Your program must compile successfully when calling “make”.
 - (b) The program must be callable as specified above.
 - (c) The program must compile without errors or warnings on our computers in the LSS CIP pool with the following g++ compiler flags (you can make use of C++11 features):

```
-std=c++0x -Wall -Wextra -Wshadow -Werror -O3 -DNDEBUG
```

Additional compiler flags are optional.

The reference compiler is GCC version 4.9.2 on the CIP Pool computers. You can check the version by typing “g++ -version”. You can load GCC 4.9.2 by typing “module load gcc/4.9.2”.

- (d) The program must be compiled with *likwid* support by default and must output the elapsed wall clock time the actual computation of the multiplication takes. You may have to load the corresponding module by typing “module load likwid” for likwid to work,. In order to determine the time it takes for your algorithm to compute the matrix-matrix multiplication, you must use the Timer class (“Timer.h”) provided on the course website. You only have to measure the time of your matrix-matrix multiplication, you do not have to measure the time for loading the matrices from disk and storing the result back to disk.
- (e) The solution should contain well commented source files, a appropriate Makefile that satisfies all the conditions specified above, instructions how to use your program (e.g. in form of a README file), and a PDF file with performance graphs. When submitting the solution, remove all temporary files from your project directory and pack it using the following command:

```
tar -cjf ex01_groupXX.tar.bz2 ex01_groupXX/
```

where XX stands for your group number and ex01_groupXX/ contains your solution. Then upload your solution in StudOn.

Performance Challenge

We will test your code and measure the performance. Every code is compiled with the g++ compiler version 4.9.2. The program is run several times on one of the i10cip1...12 machines. We will use the problem size 2048^2 when measuring the performance. The winning team will be awarded an extra credit point.

Beat Your Tutors!

The ultimate performance challenge: If you win the performance challenge and also beat the time of “/software/opensuse/siwir/matmult”, you may choose to present your implementation in the tutorial. This means you do not have to become acquainted with another topic for the tutorial! On top of that, there will be another prize that will be disclosed once your program was demonstrated to beat our implementation.

Credits

In this assignment, points are awarded as follows:

1. You are guaranteed to receive at least one point if your program correctly performs the above tasks and fulfills all of the above requirements. The correctness will be assessed by checking whether your program matches the reference outputs. Make sure that your program also works with non-square matrices! Submissions with compile errors will receive zero points! The computers in the LSS CIP pool act as reference environments.
2. Two points are awarded if, additionally, you 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, one for L2 bandwidth, one for the L2 cache miss rate, one for the double precision FLOPS, and one for runtime, with the matrix size in \log_2 scale on the abscissa. Show in each diagram 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.