02614 High-Performance Computing Jan 2023

# Assignment 1: Matrix Multiplication

Your report (PDF) and the required sources (ZIP) must be handed in electronically!

Deadline: latest on Friday, January 6, 2023 at 16:00!

## Background

The BLAS (Basic Linear Algebra Subroutines) is a collection of subroutines which are used as building blocks for linear algebra software. For instance, LAPACK and ScaLAPACK use BLAS.

The BLAS exists in a so-called "model implementation" written in FORTRAN 77. A good source of numerical software is NETLIB at `http://www.netlib.org`.

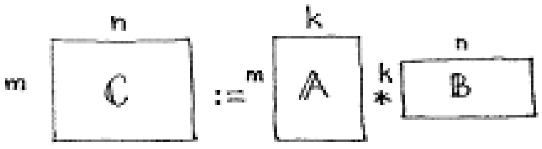
Most computer vendors deliver a BLAS/LAPACK library that is particularly optimized for their own hardware. There are also different Open Source implementations available, and one of the most common ones is ATLAS (Automatically Tuned Linear Algebra Software).

One of the most important subroutines in the BLAS library is the general matrix-matrix multiplication routine called `DGEMM`, where D stands for DOUBLE PRECISION, GE for "general" and MM for "matrix matrix." This routine is the core routine in the HPC Linpack benchmark, that is the basis for the TOP500 list of the fastest computers in the world.

## The Assignment

In this assignment, you will have to develop a library of functions, that all carry out matrix-matrix multiplication, as specified below. As a part of this assignment, we provide a framework with a driver program on DTU Learn, that can call your routines and print the timings, etc. Your library functions will be evaluated with the same tools, so you have to write your library according to the specifications provided.
For more details about the interface specifications, requirements, different coding styles, etc, see the documentation coming with the tools on DTU Learn.

I. - As a starting point, write a function, matmult_nat(), that performs a matrix-matrix multiplication with double precision matrix operands of suitable, but otherwise arbitrary, shape.



The function takes six arguments: m, n, and k, and the three matrices A, B, and C in row-major format - for the details see the provided specifications. This is very similar to the task in the lab exercises from the day(s) before, and with the help of the provided driver you will be able to check correctness, etc.

- To fulfill the requirements, and to make it easier for the remainder of the project, e.g. comparing performance, you should wrap the call to `DGEMM()` into a function matmult_lib(), that takes the same arguments as the other functions, i.e. m, n and k as well as A, B, and C. You are free to choose either the `DGEMM()` function from the "original" BLAS library, or the CBLAS implementation, as provided by libraries like ATLAS, etc. In any case, you should document your choices of arguments, when calling the library function from your wrapper.

II. - Essentially, the matrix-matrix multiplication algorithm consists of three nested loops. In how many ways can the loops be nested?

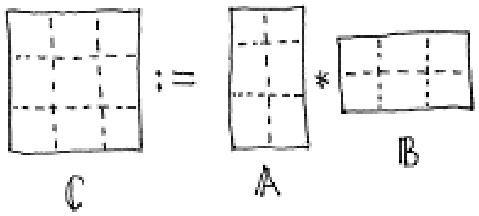
    Write a function for each of the permutations and use the three letters m, n, and k to label the versions, e.g. "matmult_mnk", "matmult_nmk", etc.

    Your version of matmult_nat() above will be identical to one of the permutations, so this one can easily be wrapped. Both should be part of the library you hand in at the end, though!

- Carry out performance experiments (using the driver program) to determine which of the permutations is the fastest, for different sizes of matrices. Compare the timings and performance numbers (a) without and (b) with (different) compiler optimizations. Report and discuss your findings.

III. Use some analysis tool, e.g. the Studio analyzer or 'perf', to measure specific characteristics, like cache hits and misses. Explain the performance differences between the different versions! Do those measurements with the tools confirm your expectations (discuss)?

IV. Write a blocked version of your matrix-matrix multiplication function, matmult_blk(), e.g. optimizing for the different cache sizes. Which of the versions from above should be your starting point (explain)? Does blocking improve the performance, and where do you expect and get the largest effect?



It is necessary to experiment with the block size bs for a given set of m, n, and k, in order to find an approximate optimum (this is a drawback of blocking). Can it be faster than the fastest non-blocked version, and if yes, for which sizes of the block size and which matrix sizes?

**Hint:** for this part, use the best compiler options!

**Note:** All the functions in your library have to work for arbitrary values of m, n, k, and the block size bs!

On machines with a large cache size it may be necessary to use matrices of substantial size in order to ascertain and measure the performance improvements you want to try. For small matrix sizes, you may want to repeat the calculations many times to get reliable timings. You can achieve this by putting an outer loop around your code and run the calculation N times (N will of course depend on the sizes of your matrices, i.e. m, n, and k). The driver tools provided have already implemented this mechanism — see the README and the description below for more details.

## Hints

1. Write the code for all required functions in the library, and do your experiments. Upload the code as well as a compiled version of the library as a ZIP archive, together with your report as a PDF file, i.e. there are **2 files** that should be uploaded: a ZIP file and a PDF file!

2. Your report should have 'theory', 'implementation/experiments' and 'results/discussion' parts for each of the sub-problems, to illustrate the flow in your working and learning process. This report is not a 'thesis' work!

3. Small pieces of code in the report are helpful to illustrate - and make it easier to understand - what you have done.

4. When using figures/tables/illustrations to show results (always nice), remember to explain what they show in the text!

5. Make sure that you answer all questions in the assignment!

6. Notice that relevant numerical experiments (runtimes, cache misses, block sizes, etc.) for large matrix sizes may take a long time to execute! However, you can reduce the time by choosing your matrix sizes in a clever way, taking your knowledge about the different cache sizes into account!

7. Make sure that you carry out sets of experiments on the same machine, to be able to compare the results. On the Linux machines, you can use the `less /proc/cpuinfo` or `lscpu` commands to get information about cache sizes, clock speed, etc. Please report those numbers in your report, as well as the compiler version you have used. It will be beneficial to use the batch system to carry out your epxeriments, since this will give you exclusive access to the resources, and you can be sure to run on the same hardware every time.

8. The front page of your report should state the names and the study numbers of all group members.

9. The addendum to the report must contain an overview of who has the main responsibility for the different parts/tasks of the assignment. See the template provided.

## Goals

During this assignment, you will

1. learn how to write efficient code
2. make use of and test the effect of compiler optimizations
3. apply tuning techniques
4. use modern analysis tools
5. learn how to interface with standard libraries
6. learn how to write a library, given an interface specification
7. analyze and document your findings in a report

## Technicalities

- Please start using the driver tools provided on DTU Learn as soon as possible in the project. This will save you a lot of time, since you do not need to write your own helper functions (e.g. allocation, timing, etc) and you can concentrate on working on the essential parts of the assignment.
- To link to the CBLAS version of ATLAS on the Linux machines, you can use the options `-L/usr/lib64/atlas -lsatlas`. If you are programming in C++, you will have to wrap the inclusion of the cblas.h header file with 'extern "C"', to avoid name mangling.
- Some libraries switch automatically to a parallel mode (multi-threaded), when executed on a multi-core computer. Please check this, since this can make a comparison of the results difficult. If you use the DTU HPC setup, this is taken care of!

**EXAMPLE**

## Addendum: Individual responsibilities in this assignment

The following table shows who has the main responsibility for the different parts in this project:

| **Part** | **studyno.** | **Name** |
|---|---|---|
| I. | s800002 | Bjørn Bjørnsen |
| II. | s800007 | Gris Grisen |
| III. | s800005 | Fugl Mejse |
| IV. | s800004 | Mus Musen |