OpenMP assignment Parallel Computing: Matrix multiplication

Karl Meerbergen, Martin Diehl, Emil Løvbak, Sahar Chehrazad, Edgar Rios Soltero

November 23, 2022

1 General information

Matrix products are a key numerical kernel in scientific computing. Such multiplications form a large part of the computational work in codes for, e.g., data-science, control and simulation. In this assignment we will make use of OpenMP to parallelize the multiplication of dense matrices. We will work with full matrices, as well as matrices containing some special structure.

As a starting point you can find two sets of files on Toledo, one implemented in `C++` and another implemented in `Fortran 90`. You are free to choose which set of files you want to work with. Each set of files consists of two files, one containing a set of implemented serial matrix products and another containing code for performing timings and checking computational results. When discussing your results it may be relevant to consider that the C++ implementation stores full matrices as a set of rows, while the Fortran implementation stores them as a set of columns. (Note that this information is only relevant if you have already followed a course on scientific software development or computer architectures.)

2 Parallelization with OpenMP

Use OpenMP pragmas to parallelize the various matrix products. Although you will not need to write much code, you should take some time to think about potential pitfalls that can occur in shared memory programming. We suggest that you follow the following steps and make notes on your findings at each step:

- Start with the diagonal product. Which loops should you parallelize? Think about which variables you have and how they are shared between threads.
- Now try the full product on the different loops what do you observe?
- An efficient implementation of matrix multiplication makes use of blocking. That is to say, that we rewrite the matrix product $C = AB$ as

$$\begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1N} \\ C_{21} & C_{22} & \cdots & C_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ C_{N1} & C_{N2} & \cdots & C_{NN} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1} & A_{N2} & \cdots & A_{NN} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1N} \\ B_{21} & B_{22} & \cdots & B_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ B_{N1} & B_{N2} & \cdots & B_{NN} \end{bmatrix}.$$

In this notation we have, e.g., that $C_{11} = A_{11}B_{11} + A_{12}B_{21} + \cdots + A_{1N}B_{N1}$, with each of the products $A_{1i}B_{i1}$ is a matrix product. If size the sub-blocks of the matrices is chosen well, then a sub-block of A, B and C should simultaneously fit in cache-memory. Parallelize the blocked implementation. What loops do you select for parallelization? You should ensure that the result you compute is consistent. Do you need to take any extra care in order to ensure this? Compare the speedup compared to the non-blocked full matrix product. How do these two algorithms compare?

- Parallelize the triangular matrix product given what you have seen above. What issue can you encounter here that is not present in the other cases? How do you avoid this?

3 Practical details

Write out your answers to the questions above in a brief report and submit it via email to `emil.loevbak@kuleuven.be` before taking part in the exam. Your report should consist of a brief discussion of the parallelization of each of the matrix product routines. Be sure to answer the questions in the above set of bullet-points and link to concepts from the lectures/exercise sessions where relevant. There is no strict page limit, but we expect that you should be able to answer these questions in 1-2 pages. If you want to to refer to specific lines of code while answering the questions you can also attach a printed out copy of the code to you report. Note that we will not look at this code unless you reference something in your written answer to the questions.

During the exercise sessions, you will have reserved computing time on the Genius cluster and the TA's will be available for asking questions. You can also test your code on the cluster outside of these sessions, but you may be placed in a queue. If you have questions you can contact us via email at `emil.loevbak@kuleuven.be`, `sahar.chehrazad@kuleuven.be` or `edgar.rios@kuleuven.be`.