Exercise 07

1 Implement a matrix-matrix multiplication routine

Write a routine that computes a matrix-matrix product given two matrices as pointers. Your function should have the following signature:

```
void matmat(double alpha, bool trans,
            double const* a, int m, int n, int lda,
            double const * b, int k, int ldb,
            double beta,
            double * c, int ldc);
```

The semantic of your function shall be: $C = \alpha \cdot A \cdot B + \beta \cdot C$ if trans is set to false, and $C = \alpha \cdot A^T \cdot B + \beta \cdot C$ otherwise.

where A is an m-by-n matrix, B is an n-by-k matrix and C is an m-by-k matrix. lda, ldb and ldc stand for “leading-dimension of” A, B and C. You can choose to layout your matrices in either row-major or column-major order, but **choose the same for all your matrices**. Suppose you pick row-major, then lda is the number of elements between the first item of row i and the first item of row i+1. Note: for matrix A this would mean lda would have to be at least n (but can be larger).

In case of column-major, it would be the number of elements between the first element of column i and the first element of column i+1. Again, for matrix A this would then mean lda would have to be at least m.

Create your own project folder with a suitable CMakeLists.txt. Look at past exercises for orientation. Divide your project into 3 files:

1. **matmat.hpp**
   This file is supposed to contain ONLY the declaration as above. It should however contain an Include-Guard
2. **matmat.cpp**
   This file contains the implementation of your matmat function. Remember to include the matmat.hpp file.
3. **main.cpp**
   This file contains the main function that allocates test matrices A, B and C and calls your function from matmat.hpp. It should also contain timing code that both measures the time used by your matmat function, as well as print the result. It will be useful to execute your function within a loop to increase the execution time.

Make sure to compile with the highest optimization level. In addition, add the flags **-qopt-report=5 -qopt-report-phase=vec** to the line of compiler flags (if you're using the Intel compiler - google the corresponding flags for your compiler if you use another one) within your CMakeLists.txt. This will generate a vectorization report for all your loops. Inspect this report in order to improve your implementation.
Take care of alignment, strided-access and pointer aliasing. Report the effects on each of your code changes.
What change had the most impact in terms of performance?
If you work with a CPU that as AVX support: switch between SSE and AVX and measure the difference in terms of performance.
Hint: To make it easier, start with a function signature that assumes all matrices to be square and of same size. Also assume that the leading dimension is N for all matrices. This simplifies the signature to

```
void matmat(double alpha, bool trans,
            double const* a, int N,
            double const * b,
            double beta,
            double * c);
```

Work your way up to the original formulation in small increments.