Otto-von-Guericke-University Magdeburg
Max Planck Institute for Dynamics of Complex Technical Systems
Computational Methods in Systems and Control Theory

Dr. Jens Saak, Dipl.-Math. Martin Köhler
Website: http://www.mpi-magdeburg.mpg.de/csc/teaching/18ws/sc1/

Winterterm 2018/2019

Scientific Computing 1
11th Homework

Handout: 12/21/2018 **Return:** 01/11/2019

Exercise 1: **(12 Points)**

New Year's Challenge

Again, we consider the generalized matrix-matrix product $C \leftarrow \alpha AB + \beta C$, where $\alpha, \beta \in \mathbb{R}$ and $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$ and $C \in \mathbb{R}^{m\times p}$. After handling memory related issues and the influence of the loop orders in the naive implementation it is now time to start a challenge. As seen in the lecture the matrix-matrix product plays also an important role during the LU decomposition (in terms of a rank-k update) and many other applications. Therefore, it is necessary to have a fast implementation and in this way we start the following challenge:

> **The person or the team who develops the fastest matrix-matrix product gets an awesome award.**

The following rules apply to the challenge:

- The code must not involved any parallelization construct, i.e., it only have to run on a single CPU core.
- The code must be written in C and can be optimized using everything which is supported by the gcc 5.5/6.3/7.3 compiler (implementation tricks, non ANSI conform C code, Compiler options, ...).
- The usage of BLAS is only allowed to check the result.
- The function signature must be

```
void gemm_opt(int m, int p, int n, double alpha,
     double *A, int lda, double *B, int ldb,
     double beta, double *C, int ldc)
```

- There are no restrictions to the matrix sizes.
- **It is up to you which strategies to chose in order to get a fast implementation but each optimization needs to be explained.**
- A `Makefile` needs to be provided in order to compile the code with your specific compiler flags.

Benchmark setup:

- The final comparison will be done on a single core of a Intel Xeon CPU Silver 4110 with 2.1GHz with 32kB L1 cache, 1MB L2 cache, and 11MB L3 cache. The used compiler will be one of the above mentioned and the best case counts. If your code relies on features only avialable in a specific compiler version you have to indicate this.
- The following problem setups need to be tested:
  a.) square matrices $m = n = p \in \{100, 500, 1000\}$,
  b.) rank-32 updates $m = p \in \{100, 500, 1000, 2000\}$ and $n = 32$,
  c.) rank-64 updates $m = p \in \{100, 500, 1000, 2000\}$ and $n = 64$.

Hint: A basic time measuring code can be found on the lecture's website.

Exercise 2: **(4 Points)**

The LU decomposition is the main technique to solve moderate size linear systems and has an asymptotic flop count of $\frac{2}{3}n^3$ flops. Beside the classical scalar implementation nowadays block implementations with a blocking parameter r are used to exploit modern computer architectures.

a.) Show that the asymptotic flop count for the block LU decomposition is invariant under the chosen blocking parameter r.

b.) Compare the exact flop count (including the lower order terms) of the unblocked and the block implementation.

Exercise 3: **(3 Points)**

Let $V, W \in \mathbb{R}^{n \times k}$, $k < n$, be two matrices of rank k. Show that

$$A = VW^T \quad \in \mathbb{R}^{n \times n}$$

has exactly rank k.

Exercise 4: **(3 Points)**

Consider the following matrix vector product

$$y = \underbrace{\left(A^T Z Z^T + Z Z^T A - Z Z^T B B^T Z Z^T + C^T C\right)}_{\mathfrak{A}} x$$

with $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times q}$, $C \in \mathbb{R}^{r \times n}$ and $Z \in \mathbb{R}^{n \times p}$. Assume that $p \ll n$, $q \ll n$ and $r \ll n$ holds.

a.) How many floating point operations are necessary to compute y naively?

b.) Rearrange the evaluation such that it takes less floating point operations.
Hint: It is necessary to introduce some extra variables.

Overall Points: 22