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/16ws/sc1/

Winterterm 2016/2017

Scientific Computing 1
9th Homework

Handout: 12/16/2016 **Return:** 01/04/2017

Exercise 1: (6 Points)

a.) Let $A \in \mathbb{R}^{n\times n}$ be a matrix and $A_{ik} \in \mathbb{R}^{(n-1)\times(n-1)}$ be the submatrix which is created by removing the i-th row and the k-th column of A. We can compute the determinant of the matrix by the recursion formula

$$\det(A) = \sum_{i=1}^{n} (-1)^{i+k} a_{ik} \det(A_{ik}).$$

Derive a recursion formula for the number of necessary floating point operations to compute $\det(A)$. Approximate the computation time for the determinant of a 100×100 matrix on a current CPU. Assume that the CPU has a peak performance of 300 GFlops/s ($300 \cdot 10^9$ floating point operations per second).

b.) Show that the determinant can be computed using the LU decomposition as $\det(A) = \prod_{i=1}^{n} u_{ii}$ Compare the computational effort with the one from a.).

Exercise 2: (6 Points)

In many cases it is not necessary to compile and link the whole LAPACK library to a program, e.g., if one only needs a single driver routine from it.

a.) Search on http://www.netlib.org/lapack/double/ for a driver which computes the eigenvalues and eigenvectors of a general matrix and download it together with its dependencies.

b.) Write a `Makefile` which creates a small static library called `liblapack_pocket.a` containing the eigenproblem solver and its dependencies.

c.) Write a small C program which uses this library to compute all eigenvalues and eigenvectors of

$$A = \begin{pmatrix} -3 & 8 & -2 & 1 \\ 6 & -4 & 1 & 5 \\ 2 & 5 & -8 & 8 \\ 1 & 5 & -8 & -7 \end{pmatrix}$$

Hint: Subroutines from BLAS are not automatically included as dependencies in the download. That means either BLAS needs to be linked to the program, or the corresponding files need to be downloaded separately.

Exercise 3: **(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 4.8.5 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 E5-2640 v3 with 2.60GHz with 32kB L1 cache, 256kB L2 cache, and 20MB L3 cache. The used compiler will be gcc 4.8.5.
- 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 $n = 64$.

Overall Points: 24