

High Performance Computing (Master) in WS22

Exercise 2: one core programming, performance limitations, BLAS.

Deadline: Nov 29, 2022, 16:00

Status:

14. Dezember 2022, 16:21

Supervisor: Prof.Dr. G. Haase,

gundolf.haase@uni-graz.at

First, we have to provide a few basic benchmarks examples (A)-(E):

(A) The inner product of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$:

$$\langle \mathbf{x}, \mathbf{y} \rangle := \sum_{i=0}^{N-1} x_i \cdot y_i$$

Use $x_i := (i \bmod 219) + 1$ and $y_i := 1.0/x_i$ as test data. (Why !?)

(B) Matrix-Vector product using an $M \times N$ matrix A and vectors $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{b} \in \mathbb{R}^M$:

$$\begin{aligned} \mathbf{b} &:= A \cdot \mathbf{x} \\ b_i &:= \sum_{j=0}^{N-1} A_{i,j} x_j \quad \forall i = 0, \dots, M-1 \end{aligned}$$

Use $A_{i,j} := ((i+j) \bmod 219) + 1$ and $x_j := 1.0/A_{17,j}$ as test data. (Why !?)
See example code¹ and docu².

(C) Matrix-Matrix product: $C_{M \times N} := A_{M \times L} \cdot B_{L \times N}$

$$C_{i,j} := \sum_{k=0}^{L-1} A_{i,k} \cdot B_{k,j} \quad \forall i = 0, \dots, M-1, \quad \forall j = 0, \dots, N-1$$

(D) Evaluation of a polynomial function of degree p with given coefficients $a_k, k = 0, \dots, p$ for a vector $\mathbf{x} \in \mathbb{R}^N$, i.e., $\mathbf{y} := \text{poly}_p(\mathbf{x}) \in \mathbb{R}^N$ with

$$y_i := \text{poly}_p(x_i) = \sum_{k=0}^p a_k \cdot x_i^k \quad \forall i = 0, \dots, N-1$$

¹http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Examples/intro_vector_densematrix.zip

²http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Examples/intro_vector_densematrix/html

(E) **Alternative i):**

Solve a sparse system of linear equations $K\underline{u} = \underline{f}$ resulting from a finite element discretization via the Jacobi iteration (code³ and docu⁴).

$$\underline{u}^{k+1} = \underline{u}^k + \omega D^{-1} \left(\underline{f} - K \cdot \underline{u}^k \right) \quad k = 0, 1, 2, \dots$$

with $\underline{u}^0 = 0$, the right hand side $\underline{f}$ and the positive definit and symmetric system matrix K .

See §4 – §5 of an old exercise⁵ and [§6.2.1 in Haase at SIAM⁶] for details.

(F) **Alternative ii):**

Solving a system of equations originating from time fractional PDEs in 1D. The challenge consists in solving a block tridiagonal system with large square matrices as blocks.

$$\begin{pmatrix} M & & & & \\ -B & A & -B & & \\ & -B & A & -B & \\ & & -B & A & -B \\ & & & & M \end{pmatrix}_{n_b \times n_b} \cdot \underline{u} = \underline{f} \quad (1)$$

Each block matrix M, A, B has dimensions $n_t \times n_t$.

System (1) is solved for $n_b := 2^k + 1$ via cyclic reduction [§4.4.1 in Haase at Teubner⁷], see code⁸ and docu⁹.

³http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Codes/seq/jacobi_oo_stl.zip

⁴http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Codes/seq/jacobi_oo_stl/html

⁵https://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Codes/par/mpi_course.pdf

⁶https://imsc.uni-graz.at/haasegu/Lectures/Master_HPC/textbook.pdf

⁷https://imsc.uni-graz.at/haasegu/Lectures/Master_HPC/trd-haas.pdf

⁸http://imsc.uni-graz.at/haasegu/Progs/gh_hack.zip

⁹http://imsc.uni-graz.at/haasegu/Progs/gh_hack/html

1. Measure the achieved floating point performance and memory bandwidth of your test system by compiling and using the codes¹⁰ for stream and flops. Have a look at the first lines in the source code for compiling options. Benchmarking is science by itself - we need the above codes to have some numbers in our hands. (1 Pkt.)
You can use additionally the precompiled benchmark codes from Alexander Yee¹¹. You will probably need the executable for Haswell (Intel-CPU) or for Zen (AMD-CPU).

2. Some counting and calculating for benchmarks (A)–(D): (1 Pkt.)
- Calculate the amount of memory needed for your data as function of the data dimensions M, N, L, p depending on the chosen data type.
 - Calculate the number of floating point operations ($+, -, *, /$ count as one operation) as function of the data dimensions M, N, L, p .
 - Express the number of Read/Write operations from/to memory as function of the data dimensions M, N, L, p and the chosen data type.

3. Implement benchmarks (A)–(D) in C++ as **separate functions** where all data will be transferred via the parameter list, i.e., no global data are allowed. The data types for **double precision** should be preferred, but you might use also template functions. (4 Pkt.)
In order to simplify code development and benchmarking you are encouraged to use the provided template for the scalar product¹²

After extracting the files you have to type

```
make run
```

into your (LINUX-) Terminal to compile, link and run the code.

4. Measure the runtime of your implementations. Calculate the achieved double precision performance in GFLOPS and the achieved memory bandwidth in GiB/sec for your implementations. (1 Pkt.)
Take care that you use compiler options wrt. optimization, i.e., "-O0" versus "-O3" and more advanced options.
Chose the data dimensions for each benchmark such that the code runs at least 10 sec. (Why?!), i.e.. If the runtime for (A) is too low although you use already 50% of your memory than you have to increase NLOOPS in the template accordingly. The memory required to store your data should be approximately 10–100 times larger than your largest cache.

5. Some special tasks: (2 Pkt.)
- (a) Measure the performance in (A) by replacing the call to $\langle x, y \rangle$ by a call to
- $$\|x\| := \sqrt{\sum_{i=0}^{N-1} x_i^2}.$$
- What do you observe? Why?

¹⁰<http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Examples/stream.zip>

¹¹<https://github.com/Mysticial/Flops/tree/master/version3>

¹²http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Codes/seq/skalar_stl.zip

- (b) Implement a version of the scalar product that avoids round-off errors by using the Kahan¹³ summation. Check the run time.
- (c) Assume row-wise access to matrix A in (C). Compare performance issues for row-wise access of A versus column-wise access of A .
Is there a chance to accelerate also the column-wise access function?
6. Write additional functions for (A)-(C) that use the cBLAS¹⁴ library¹⁵, especially the calls DOT, GEMV, GEMM, see the general reference card¹⁶ or the cblas description¹⁷, see the example for cblas_dgemv¹⁸. Measure run time and calculate GFLOPS and GiB/-sec. (4 Pkt.)

Hints:

- Take care for the Leading Dimensions (LDA, LDB) in the parameter list for a dense matrix in BLAS. See the example for DGEMM in stackoverflow¹⁹ at IBM²⁰ and at Intel²¹.
 - The LAPACK²² library is a superset of the BLAS routines based on FORTRAN. Note the additional parameter for cblas_dgemv from cBLAS²³ in comparison to dgemv_ from BLAS²⁴.
The LAPACK interface including BLAS requires a columnwise storage of the matrix entries. Using the extension LAPACKE²⁵ allows an optional rowwise storage of the matrices similar to cBLAS.
 - Install the libraries in Ubuntu via

```
sudo apt-get install libopenblas-base libopenblas-dev
                                liblapack-dev liblapacke-dev
```

or in archlinux via `sudo pacman -S openblas; sudo pacman -S lapacke`
 - Use the example with class Blas_DenseMatrix (code²⁶ and docu²⁷) as a start.
- (**) What is the best BLAS available? Atlas²⁸ with Fortran-Blas and cblas²⁹, uBLAS³⁰ (C++ boost) or the MKL-library by INTEL?

¹³https://en.wikipedia.org/wiki/Kahan_summation_algorithm

¹⁴<http://www.netlib.org/blas/faq.html>

¹⁵ubuntu: libatlas-dev, libatlas-base-dev

¹⁶<http://www.netlib.org/blas/blasqr.pdf>

¹⁷<https://software.intel.com/en-us/mkl-developer-reference-c-blas-and-sparse-blas-routines>

¹⁸http://www.netlib.org/lapack/explore-html/d1/dff/cblas__example1_8c_source.html

¹⁹<http://stackoverflow.com/questions/28654438/matrix-vector-product-with-dgemm-dgemv>

²⁰http://www.ibm.com/support/knowledgecenter/SSFHY8_5.2.0/com.ibm.cluster.essl.v5r2.essl100.doc/am5gr_hsgemm.htm

²¹<https://software.intel.com/en-us/node/429920>

²²<http://www.netlib.org/lapack/>

²³<https://software.intel.com/en-us/mkl-developer-reference-c-cblas-gemv>

²⁴http://www.netlib.org/lapack/explore-html/d7/d15/group__double__blas__level2_gadd421a107a488d524859b4a64c1901a9.html

²⁵http://www.netlib.org/lapack/explore-html/dir_01b0e9dedaacaad221a964f5affeb80.html

²⁶http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Examples/densematrices_libs.zip

²⁷http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Examples/densematrices_libs/html

²⁸/usr/share/doc/libatlas-doc

²⁹http://www.gnu.org/software/gsl/manual/html_node/GSL-CBLAS-Library.html

³⁰https://www.boost.org/doc/libs/1_80_0/libs/numeric/ublas/doc/index.html

7. Solve a dense system of equations via factorization LAPACKE_dgetrf³¹ of matrix $A_{n \times n}$ and application of the factorized matrix LAPACKE_dgetrs³² to multiple right hand sides $b_{n \times n_{rhs}}$. (4 Pkt.)

- Check for the correct result.
- How does the solution time per right hand side depend on n_{rhs} for a fixed $n \in \{500, 1000, 2000\}$?
- One suggestion for initializing the matrix entries:

$$M_{i,j} := \begin{cases} \frac{1}{|i-j|^2} & \text{if } i \neq j \\ 4 & \text{if } i = j \end{cases}$$

8. Run (E), the Jacobi code from directory *seq/jacobi* . (1 Pkt.)

9. **Alternatively to 6.-8.:** Run and improve (F) (9 Pkt.)

14. Dezember 2022

³¹http://www.netlib.org/lapack/explore-html/d2/d96/lapacke__dgetrf_8c_a285c069fa65d2b9954737240e0779889.html

³²http://www.netlib.org/lapack/explore-html/d8/d54/lapacke__dgetrs_8c_a044782d17dd9592ab7d2b5b4f5c0f7a7.html