An introduction to numerical methods with BLAS

Georgios A. Kafanas

High Performance
Computing &
Big Data Services

hpc.uni.lu

hpc@uni.lu

@ULHPC

LUXEMBOURG
LET'S MAKE IT HAPPEN

UNIVERSITÉ DU
LUXEMBOURG

1

Outline

- Setting up a system to compile a BLAS library and link it with executables
- Data representation for vector and matrices
- The interface of BLAS
- Effects of caches and cache aware programming
- Resources for testing and benchmarking

Not all evaluations perform the same!

Consider a simple matrix-vector multiplication:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix}. \tag{1}$$

The evaluation can be reduced with either 2 dot product or 2 scalar-matrix product operations:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix} \\ \begin{pmatrix} 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix} \end{pmatrix},$$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix} = 1 \begin{pmatrix} 1 \\ 3 \end{pmatrix} + 2 \begin{pmatrix} 2 \\ 4 \end{pmatrix}.$$

Are there differences in the efficiency of the 2 evaluation methods?

Setting up your system to compile a BLAS library and link it with executables

Preparing your system to compile BLAS

1. Install git

Preparing your system to compile BLAS

1. Install git
2. Install build tools:
   - gcc
   - gfortran
   - CMake
   - Ninja

Compiling Netlib BLAS

Fetching the code

- Clone the git repository from the UL HPC GitLab server.

```
$ git clone ssh://git@gitlab.uni.lu:8022/hlst/seminars/blas/lapack.git ~/Documents/blas/lapack
```

- Checkout the branch with the setup for the tutorial

```
$ cd ~/Documents/blas/lapack
$ git checkout blas-tutorial
```

Compiling Netlib BLAS

Build the BLAS components of Netlib LAPACK

- Configure the build system with the preset options

```
$ cmake --preset default-config
```

- The command will create a directory 'build' with the instructions to build the software

```
$ cmake --build build/Release --target cblas
$ cmake --build build/Release --target CBLAS/install
```

- The commands will:
  - build BLAS and CBLAS components
  - install the libraries in '~/.local/netlib'
- See the 'CMakePresets.json' for more information

Linking with Netlib BLAS

Fetch the example code

```
$ git clone ssh://git@gitlab.uni.lu:8022/hlst/seminars/blas/blas-tutorial.git ~/Documents/blas/blas-tutorial
```

- Configure the examples

```
$ cd ~/Documents/blas/blas-tutorial
$ cmake --preset default-config
```

Linking with Netlib BLAS

Build the example code

```
$ cmake --build build/Release --target 00_example_ddot --verbose
```

- The output is extensive, but we can reproduce the result with the following commands:

```
$ mkdir lib bin
$ gcc -I./include -isystem ${HOME}/.local/netlib/include -O3 -DNDEBUG -o src/00_example_ddot.c.o -c src/00_example_ddot.c
$ gcc -O3 -DNDEBUG -o src/utils.c.o -c src/utils.c
$ ar qc lib/libutils.a  src/utils.c.o && ranlib lib/libutils.a
$ gcc -O3 -DNDEBUG -Wl,--no-as-needed src/00_example_ddot.c.o -o bin/00_example_ddot
-Wl,-rpath,${HOME}/.local/netlib/lib  lib/libutils.a ${HOME}/.local/netlib/lib/libcblas.so
${HOME}/.local/netlib/lib/libblas.so -lm
```

Linking with Netlib BLAS

Let's break down the compilation commands

```
$ gcc -I./include -isystem ${HOME}/.local/netlib/include -O3 -DNDEBUG -o src/00_example_ddot.c.o -c
src/00_example_ddot.c
```

- Create an object file from 'src/00_example_ddot.c'
- The file 'src/00_example_ddot.c' calls function from:
    - 'utils/h': add the location of the header file in the search path with '-I'
    - 'cbals.h': add the location of the header file in the system search path with '-isystem'
- Directories included with '-I' are searched before directories included with '-isystem'
- Headers in directories included with '-isystem' must use '<...>' notation

Linking with Netlib BLAS

Let's break down the compilation commands

```
$ gcc -O3 -DNDEBUG -o src/utils.c.o -c src/utils.c
```

- Creates an object file for the utility library from 'src/utils.c'

```
$ ar qc lib/libutils.a  src/utils.c.o && ranlib lib/libutils.a
```

- Composes the object file into a static library
  - Effectively an object file with a lookup table to easily locate functions within the library

Linking with Netlib BLAS

Let's break down the compilation commands

```
$ gcc -O3 -DNDEBUG -Wl,--no-as-needed src/00_example_ddot.c.o -o bin/00_example_ddot
-Wl,-rpath,${HOME}/.local/netlib/lib  lib/libutils.a ${HOME}/.local/netlib/lib/libcblas.so
${HOME}/.local/netlib/lib/libblas.so -lm
```

- The command links the executable 'bin/00_example_ddot'
  - **-Wl,-rpath:** Adds the location of the BLAS and CBLAS library in the executable RUNPATH
  - **-Wl,--no-as-needed:** ensures that all dependencies are loaded from the correct location
  - **-lm:** links with a mathematical library (libm) used in 'utilities.c'
- Dynamic libraries 'libblas.so' for BLAS and 'libcblas.so' and CBLAS will be required in runtime
- Static library 'libutils.a' is linked statically and will not be required in runtime

Linking with Netlib BLAS

Run the executable

- Compile the executable with the cmake build scripts:

```
$ cmake --build build/Release --target 00_example_ddot --verbose
```

- After the compilation you will be able to run the resulting executable with:

```
$ ./build/Release/bin/00_example_ddot # print help
$ ./build/Release/bin/00_example_ddot 100 2 # An example run
```

Linking with Netlib BLAS

Practical session

- Try compiling the CBLAS library
- Try compiling the executable '00_example_ddot'

Software libraries

- BLAS is a typical software library
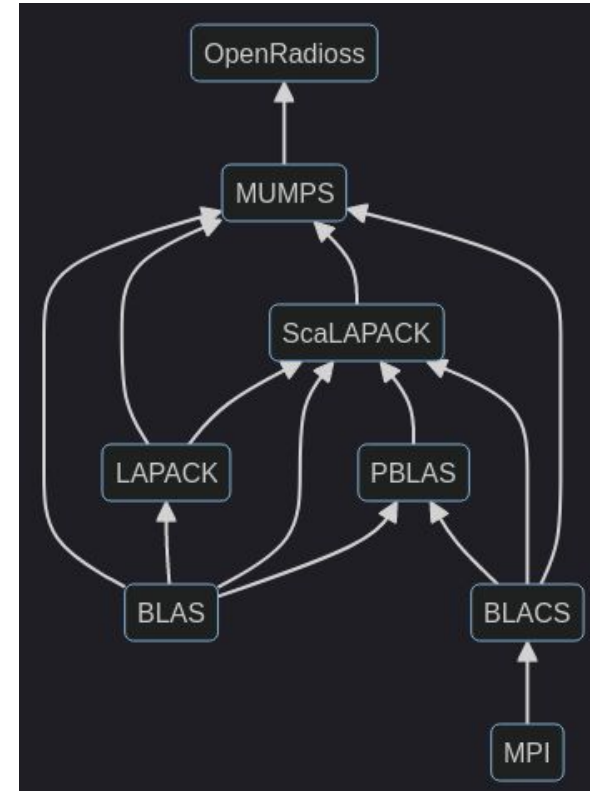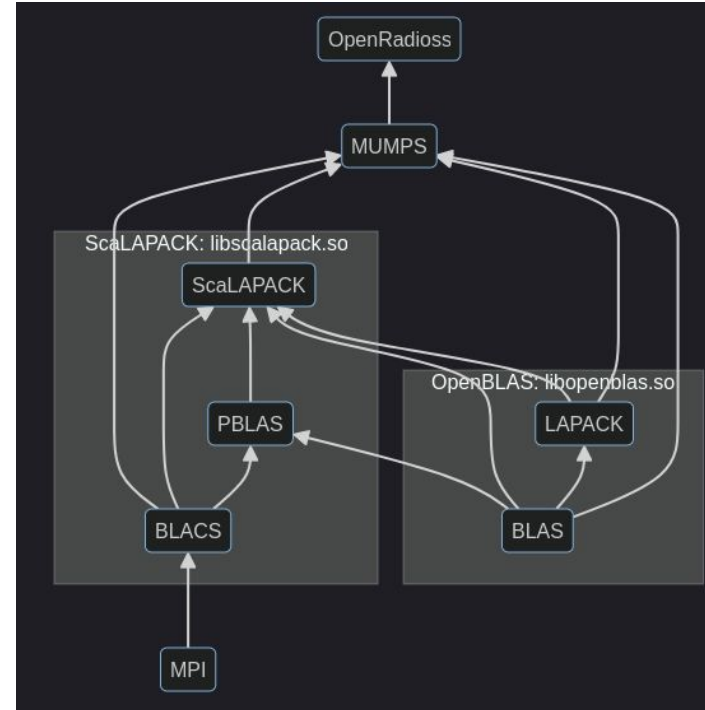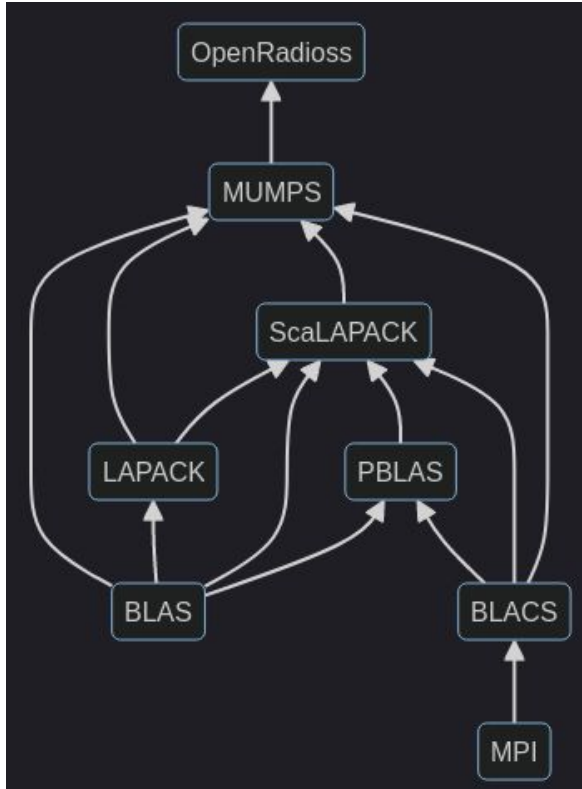- Libraries can be used in 2 forms:
    - Static
    - Dynamic
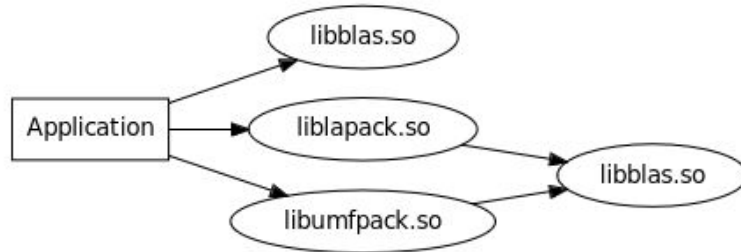
Figure 1: Shared library dependencies of an example application.
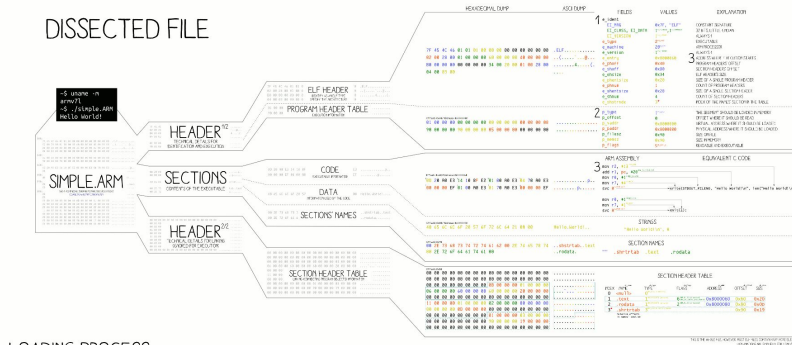
Software libraries

Tools for inspecting libraries and executables

- Does a particular 'libopenblas.so' instance implement the CBLAS interface?

Tools for inspecting libraries and executables

To investigate the shared object:

- `readelf`: display information about ELF files
  - `--all`: all sections
  - `--file-header`: information about interoperability
  - `--dynamic`: dynamically linked libraries and other information
- `objdump`: display information about objects
  - `--syms`: information for symbols (functions and variables)
  - `--demangle`: restore human readable names for objects generated from C++
- `nm`: list symbols
  - `--dynamic`: list only export symbols (only for dynamic libraries)
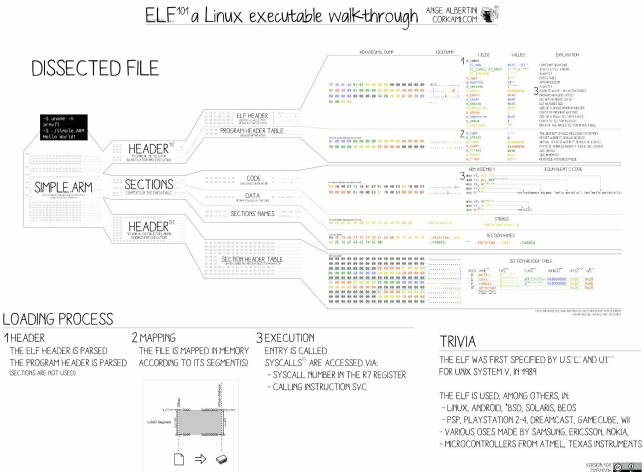
Tools for inspecting libraries and executables

Even extract information about function signatures (needs debug info, `-g`):

- Read debug info with `readelf`
  - `--debug-dump=info`
- Partially disassemble with `objdump`
  - `--disassemble`
  - `--disassemble-all`

ELF 101 a Linux executable walk-through — ANGE ALBERTINI CORKAMI.COM
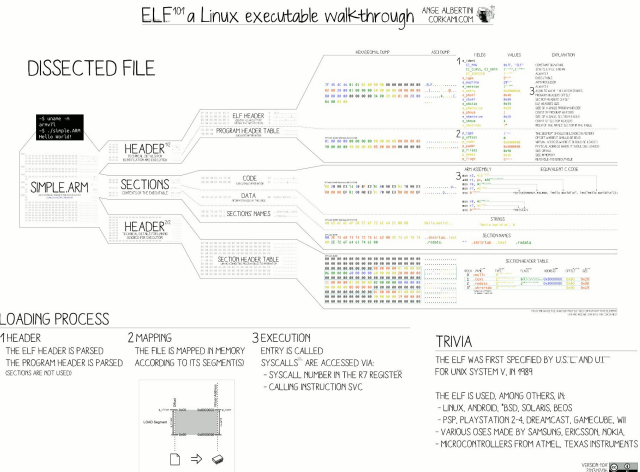
Tools for inspecting libraries and executables

Practical session

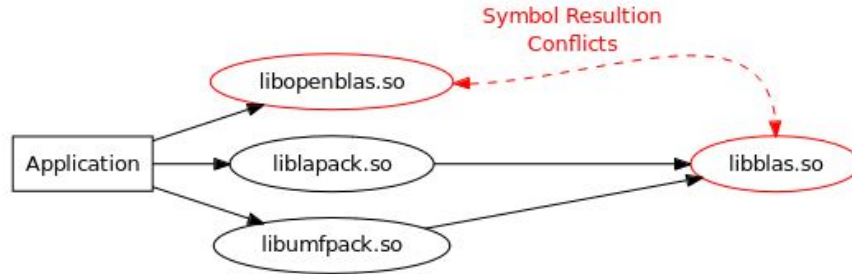- Can you break the linking? Try removing the linker option: `--no-as-needed`

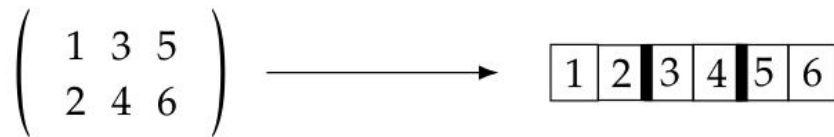Figure 2: Wrong symbol resolution after relinking the example application.

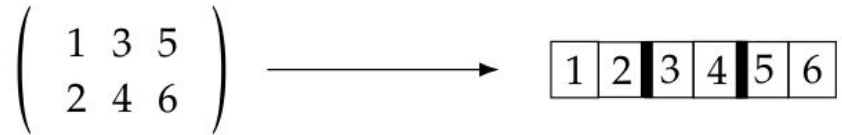# Data representation for vector and matrices

Data representation

Matrices

- Computer memory is linear
- Matrices must be linearized:

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \longrightarrow \boxed{1 \;|\; 2 \;|\; 3 \;|\; 4 \;|\; 5 \;|\; 6}$$

Data representation

structure matrix(T)
    data : T*
    m : integer
    n : integer
end structure

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \longrightarrow$$

| 1 | 2 | 3 | 4 | 5 | 6 |

- How useful is this representation?

Data representation

Example: Gaussian elimination algorithm

$$A^{(0)} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \end{pmatrix} \sim A^{(1)} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

$$A^{(0)} \equiv \begin{pmatrix} 1 & 1 & 1 & | & 1 & 2 & 2 & | & 1 & 2 & 3 \end{pmatrix}$$

$$A^{(1)} \equiv \begin{pmatrix} 1 & 0 & 0 & | & 1 & 1 & 1 & | & 1 & 1 & 2 \end{pmatrix}$$

Data representation

Example: Gaussian elimination algorithm

structure matrix(T)
    data : T*
    ld : integer
    m : integer
    n : integer
end structure

$$A^{(1)} \equiv \left(\begin{array}{ccc|ccc|ccc} 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 2 \end{array} \right)$$

$$A^{(1)} = \left(\begin{array}{c|cc} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \end{array} \right)$$

The interface of BLAS

The BLAS interface

- Operations organized by computational complexity
  - Level 1: O(n)
  - Level 2: O(n^2)
  - Level 3: O(n^3)
- BLAS supports various number types and numerical precision (first part of function names):

  - single precision (**S**) with 32-bits,

  - double precision (**D**) with 64-bits,

  - single precision complex (**C**) with 64-bits, and

  - double precision complex (**Z**) with 128-bits.

The BLAS interface

- Matrix properties are exploited to save space and reduce memory accesses:

GE:
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \longrightarrow$$

a_{11}	a_{21}	a_{31}	a_{12}	a_{22}	a_{32}	a_{13}	a_{23}	a_{33}

SY:
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{pmatrix} \longrightarrow$$

a_{11}	a_{21}	a_{31}	a_{12}	a_{22}	*	a_{13}	*	*

TR:
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{pmatrix} \longrightarrow$$

a_{11}	a_{21}	a_{31}	a_{12}	a_{22}	*	a_{13}	*	*

SP:
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{pmatrix} \longrightarrow$$

a_{11}	a_{21}	a_{31}	a_{12}	a_{22}	a_{13}

TP:
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{pmatrix} \longrightarrow$$

a_{11}	a_{21}	a_{31}	a_{12}	a_{22}	a_{13}

The BLAS interface

- The type of the matrix representation used forms the 2nd part of the name:

Algebraic properties	Storage type		
	Standard (–)	Banded (B)	Packed (P)
General (G)	GE	GB	
Symmetric (S)	SY	SB	SP
Hermitian (H)	HE	HB	HP
Triangular (T)	TR	TB	TP

The BLAS interface

- Last part is the type of the operants:
  - V: vector
  - M: matrix
- For instance:

DGEMV:

  - D: double precision
  - GE: general matrix
  - MV: matrix-vector multiplication

DGEMM:

  - D: double precision
  - GE: general matrix
  - MM: matrix-matrix multiplication

- The convention does not work always, especially for Level 1 operations
  - DAXPY: y   ax+y

Data representation & BLAS interface

Practical session

- Try exercises 01-03.
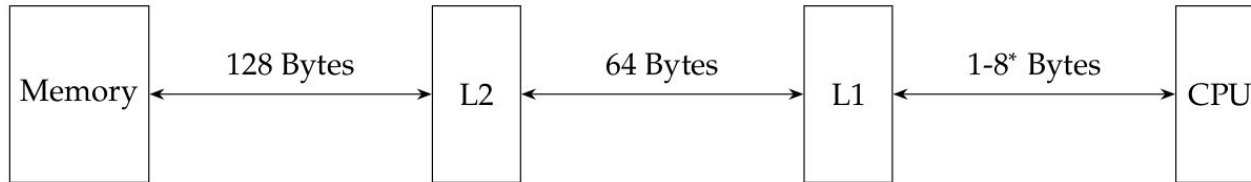- You will need to find and call the appropriate BLAS functions.

Effects of caches and cache aware programming

Effects of caching

- Direct linearization of matrices can degrade performance!
- Caches affect the speed of memory access

```
for ( int i = 0; i < n; ++i ) {

  a[i] = 0;
}
```


*up to 64 for some special SIMD instructions sets such as AVX-512

Effects of caching

- Direct linearization of matrices can degrade performance!
- Caches affect the speed of memory access

```
for ( int i = 0; i < n; ++i ) {

  a[i] = 0;
}
```
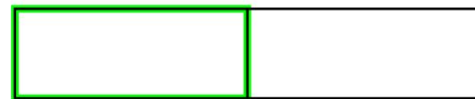
Vectorizable:

Non-vectorizable:

Effects of caching

- Direct linearization of matrices can degrade performance!
- Caches affect the speed of memory access

```
#pragma omp simd aligned(a:32)
for ( int i = 0; i < 4*n; i+=1 ) {
  a[i] = 0;
}
```
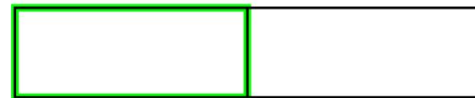
Vectorizable:

Non-vectorizable:

Effects of caching

- Direct linearization of matrices can degrade performance!
- Caches affect the speed of memory access

```
for ( int i = 0; i < n; i+=1 ) {

  a[4*i] = 0;
  a[4*i+1] = 0;
  a[4*i+2] = 0;
  a[4*i+3] = 0;
}
```

Vectorizable:

Non-vectorizable:

Effects of caching

Practical session

- Try exercises 04-05
- Demonstration of the use of 'aligned_alloc' in 'src/read_and_execute.c'
- Hint: *for exercise 05 use the function 'get_ld' of the utilities library to get the leading dimension of arrays*

Resources for testing and benchmarking

Resources for testing and benchmarking

Resources for BLAS

- Official BLAS webpage: https://www.netlib.org/blas/
- Quick reference (function list): https://www.netlib.org/blas/
- Reference BLAS implementation:
  https://www.netlib.org/lapack/explore-html/d1/df9/group__blas.html

Matrix benchmarks

- Matrix market: https://math.nist.gov/MatrixMarket/
- Matrix market exchange format reader:
  https://gitlab.uni.lu/hlst/seminars/blas/matrix-market-exchange-formats

Thank you!

Any questions?