An introduction to numerical methods using BLAS

Georgios Kafanas

GreekLUG

26 November 2023

Overview

- What are numerical libraries and how to use them?
- Functionality and uses of BLAS
- How computer architecture (caches) affect implementation of computations

Not all evaluations are the same!

Consider a simple matrix multiplication

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

Can be evaluated in 2 ways, with a dot product or a scalar-matrix product:

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

Software libraries

- BLAS is a typical software library
- Libraries can be used in 2 forms:
    - Static
    - Dynamic

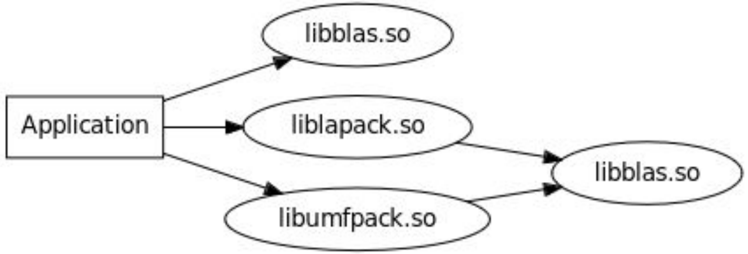
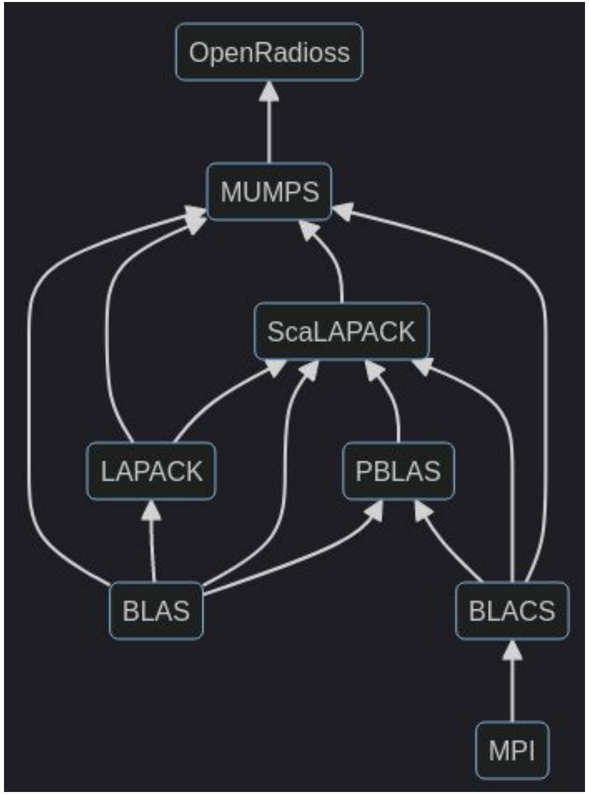
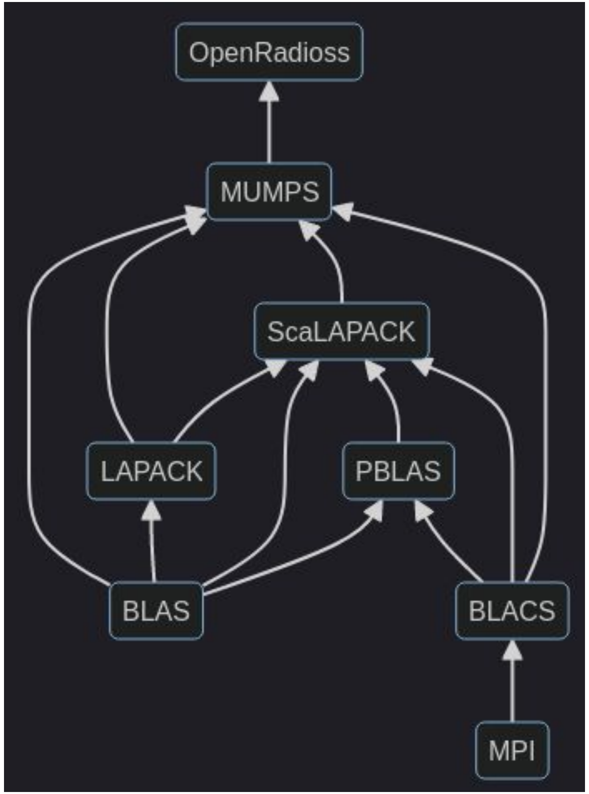
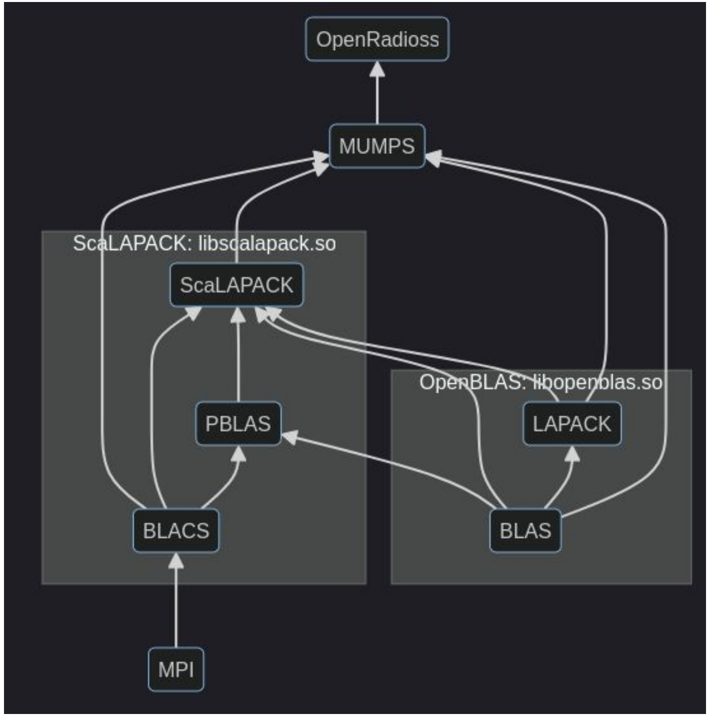


Figure 1: Shared library dependencies of an example application.



Software libraries





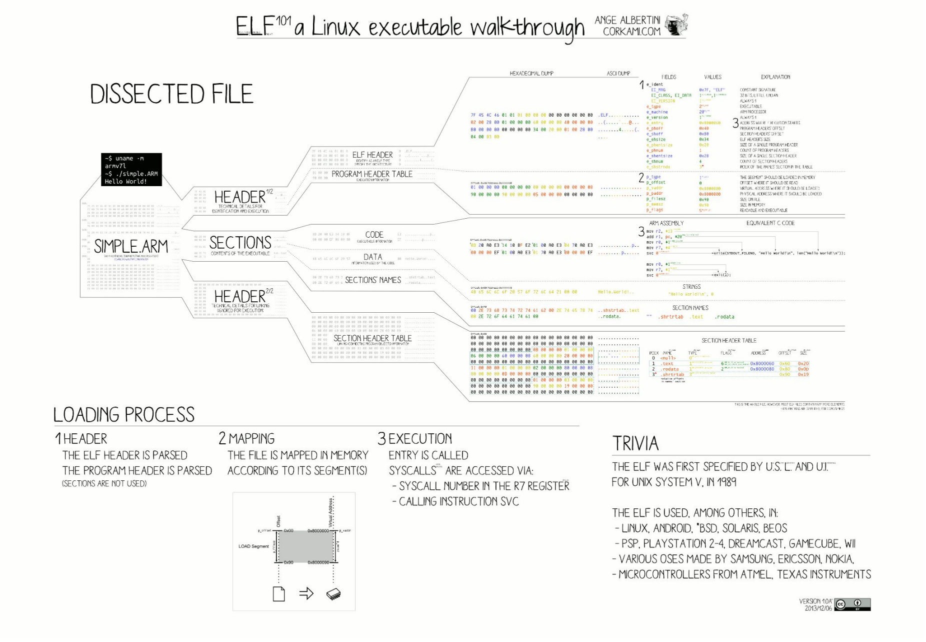
Practical session

Compile and install:

- BLAS: https://gitlab.com/greeklug/lapack/-/tree/greeklug-presentation
- Matrix-Market I/O library: https://gitlab.com/greeklug/matrix_market_exchange_formats

Tools for inspecting libraries and executables

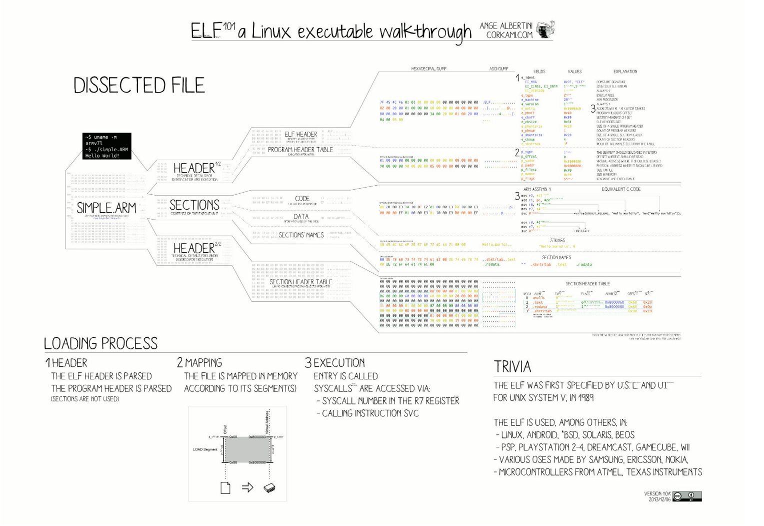
- Does this `libopeblas.so` instance implement the CBLAS interface?



Tools for inspecting libraries and executables

To investigate the shared object:

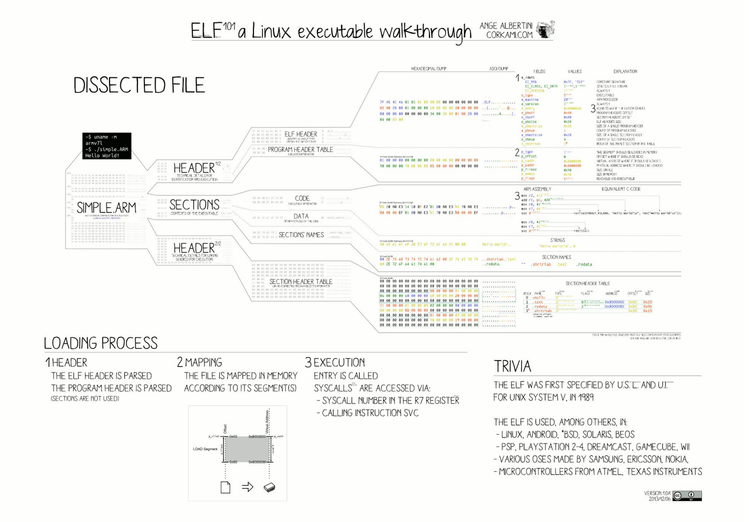
- `readelf`: display information about ELF files
  - `--all`: all sections
  - `--file-header`: information about interoperability
  - `--dynamic`: dynamically linked libraries and other information
- `objdump`: display information about objects
  - `--syms`: information for symbols (functions and variables)
  - `--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`



Practical session

- Compile the tutorial example code: https://gitlab.com/greeklug/blas-tutorial
- Call some function Matrix Market I/O
- Can you break the linking? Try removing the linker option: `--no-as-needed`

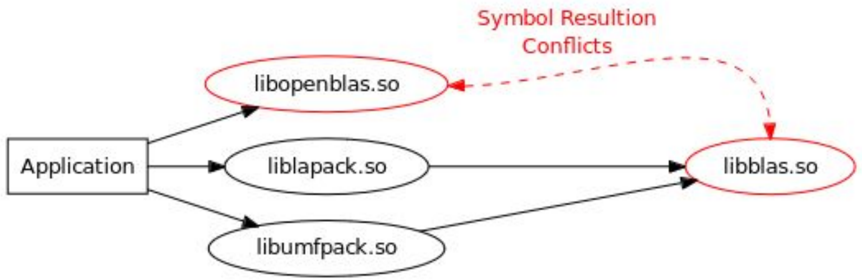


Figure 2: Wrong symbol resolution after relinking the example application.

Data representation

- Computer memory is linear
- Matrices are linearized:

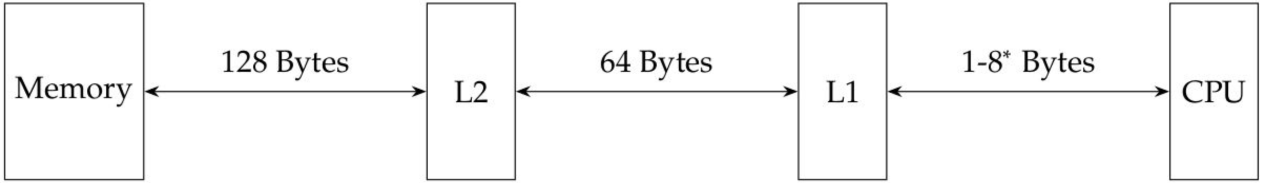
$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \longrightarrow \boxed{1}\boxed{2}\boxed{3}\boxed{4}\boxed{5}\boxed{6}$$

```c
typedef struct _dense_matrix {
  /* Data structure storing matrix A */
  double* a; // Pointer to the C array with the entries of A
  int m;     // Number of rows in A
  int n;     // Number of columns in A
} dense_matrix;
```

Caching

- Direct linearization is not sufficient for good performance!
- Caches affect the speed of memory access

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

  a[i] = 0;
}
```

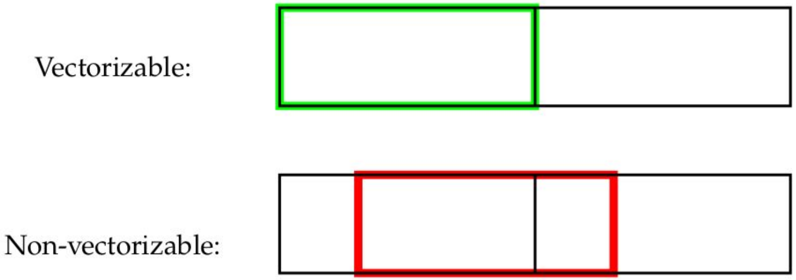


*up to 64 for some special SIMD instructions sets such as AVX-512

Caching

- Direct linearization is not sufficient for good performance!
- Caches affect the speed of memory access

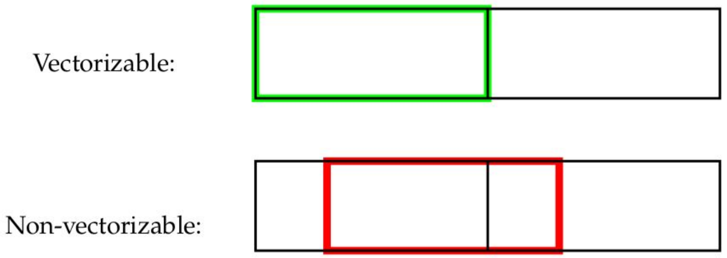
```
for ( int i = 0; i < n; ++i )
{
  a[i] = 0;
}
```



Caching

- Direct linearization is not sufficient for good performance!
- Caches affect the speed of memory access

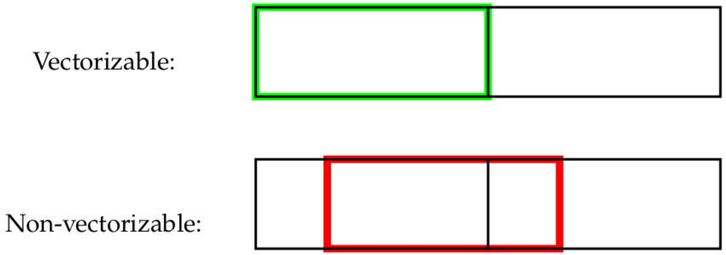
```
#pragma omp simd aligned(a:32)
for ( int i = 0; i < 4*n; i+=1 ) {

  a[i] = 0;
}
```



Caching

- Direct linearization is not sufficient for good 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;
}
```



Not all evaluations are the same!

Revisiting the simple matrix-vector product:

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

The dot product evaluation jumps across cache lines!

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

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

Caching

- Direct linearization is not sufficient for good performance!
- Caches affect the speed of memory access

```c
typedef struct _dense_matrix {
  /* Data structure storing matrix A */
  double* a; // Pointer to the C array with the entries of A
  int m;     // Number of rows in A
  int n;     // Number of columns in A
  int nzmax; // Maximum number of entries that can be stored in array a
  int lda;   // Leading dimension of the array A
} dense_matrix;
```

Practical session

- Call some function (DGEMV) of BLAS
- Try the code with aligned memory allocation!

BLAS naming conventions

- 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.

BLAS naming conventions

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

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

SY:
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{pmatrix} \longrightarrow \begin{array}{|c|c|c|c|c|c|c|c|c|}\hline a_{11} & a_{21} & a_{31} & a_{12} & a_{22} & * & a_{13} & * & * \\ \hline \end{array}$$

TR:
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{pmatrix} \longrightarrow \begin{array}{|c|c|c|c|c|c|c|c|c|}\hline a_{11} & a_{21} & a_{31} & a_{12} & a_{22} & * & a_{13} & * & * \\ \hline \end{array}$$

SP:
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{pmatrix} \longrightarrow \begin{array}{|c|c|c|c|c|c|}\hline a_{11} & a_{21} & a_{31} & a_{12} & a_{22} & a_{13} \\ \hline \end{array}$$

TP:
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{pmatrix} \longrightarrow \begin{array}{|c|c|c|c|c|c|}\hline a_{11} & a_{21} & a_{31} & a_{12} & a_{22} & a_{13} \\ \hline \end{array}$$

BLAS naming conventions

- Matrix properties are exploited to save space and reduce memory accesses.
- This forms the second part of the name:

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

BLAS naming conventions

- 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: $y \leftarrow ax+y$

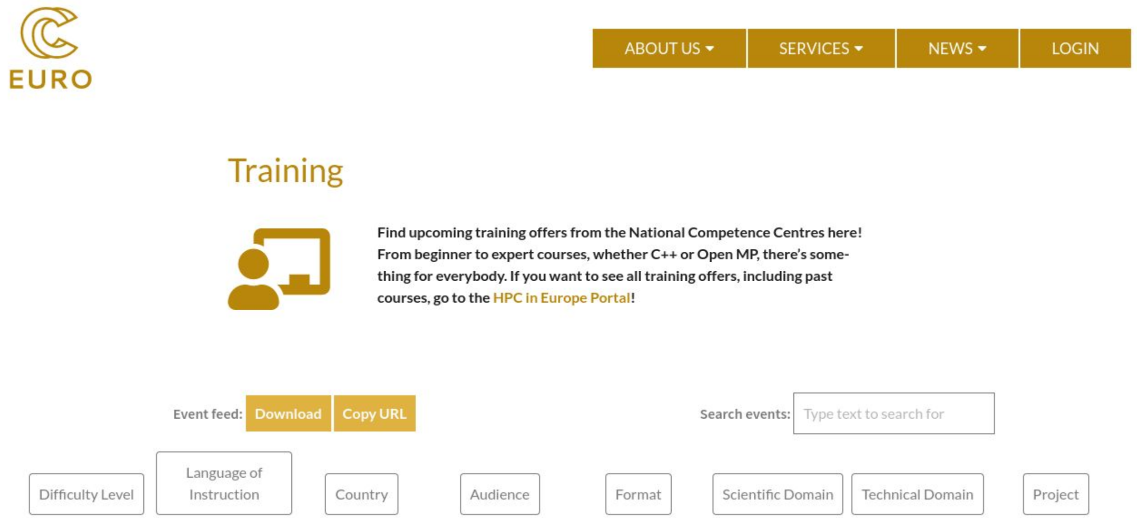
Course notes and resources

Will appear in the tutorial directory: https://gitlab.com/greeklug/blas-tutorial

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

Seminar on numerical methods coming soon in EurroCC:
https://www.eurocc-access.eu/services/training/

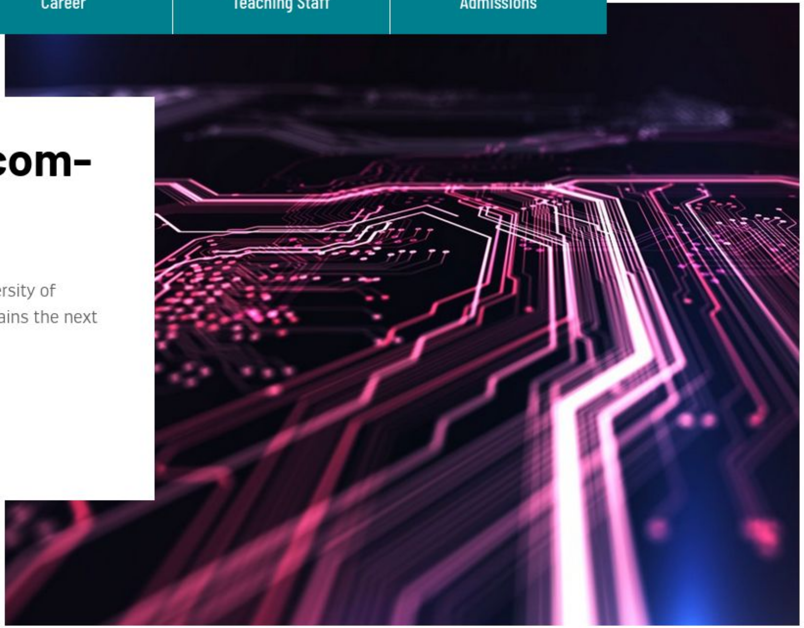


Thank you!

Study Programme

Master in High Performance Computing

Overview | Programme | Career | Teaching Staff | Admissions



Deepen your career in computing

The Master in High Performance Computing (HPC) at the University of Luxembourg is a 2-year innovative Master's programme that trains the next generation of HPC experts in Luxembourg and Europe.

Learn more | Teachers

The programme at a glance – 120 ECTS

Duration: 2 years / 4 sem

Language: EN

Admissions
EU: 1 Feb-28 Aug 2024
Non-EU: 1 Feb-30 Apr 2024

Fees: €200/sem

Format: Full time

Campus: Belval

Available places: 40

Study Programme

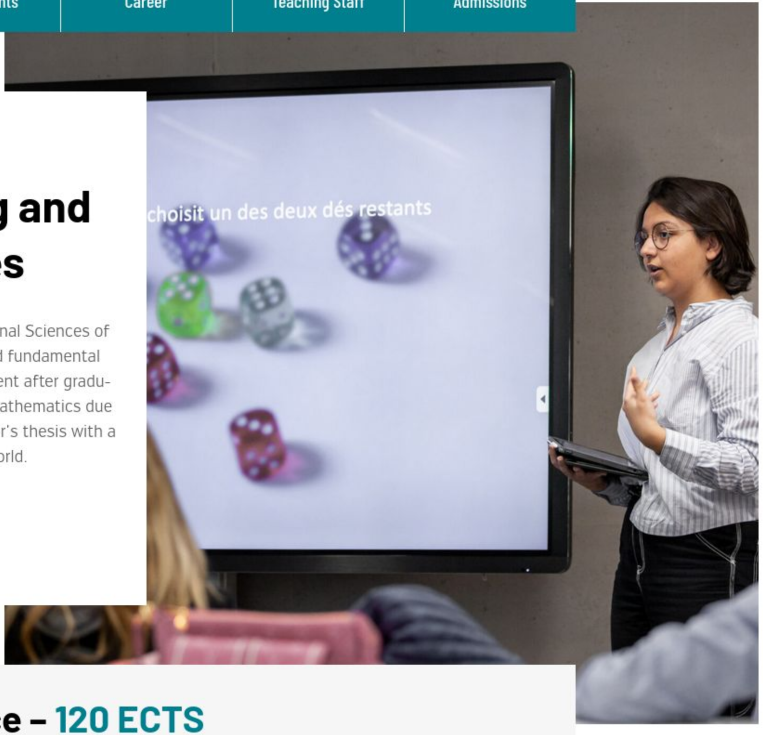
Master in Mathematics – Mathematical Modelling and Computational Sciences

Launch your career in Mathematical Modelling and Computational Sciences

The two-year track in Mathematical Modelling and Computational Sciences of the Master in Mathematics focuses on both computational and fundamental aspects of mathematics and prepares for immediate employment after graduation. You will become thoroughly acquainted with industrial mathematics due to the possibility of doing a summer internship and your master's thesis with a local company and thus require fruitful insights into the real world.

Learn more

Teachers



The Programme at a glance – 120 ECTS

Duration:
2 years / 4 sem

Teaching Languages:
EN

Admissions:
EU: 1 Feb 2023 – 31 Aug 2023
Non-EU: 1 Feb 2023 – 30 Apr 2023

Fees:
200€/ sem. (semester 1)


EUMaster4HPC
European Master
For High Performance Computing
Application
Summer School
Research & Industry
Applicants from participant countries are eligible for receiving EU funds.

- Master in High Performance Computing: https://www.uni.lu/fstm-en/study-programs/master-in-high-performance-computing/
- Master in Mathematics - Mathematical Modelling and Computational Sciences: https://www.uni.lu/fstm-en/study-programs/master-in-mathematics-mathematical-modelling-and-computational-sciences/
- European Master For High Performance Computing: https://eumaster4hpc.uni.lu/