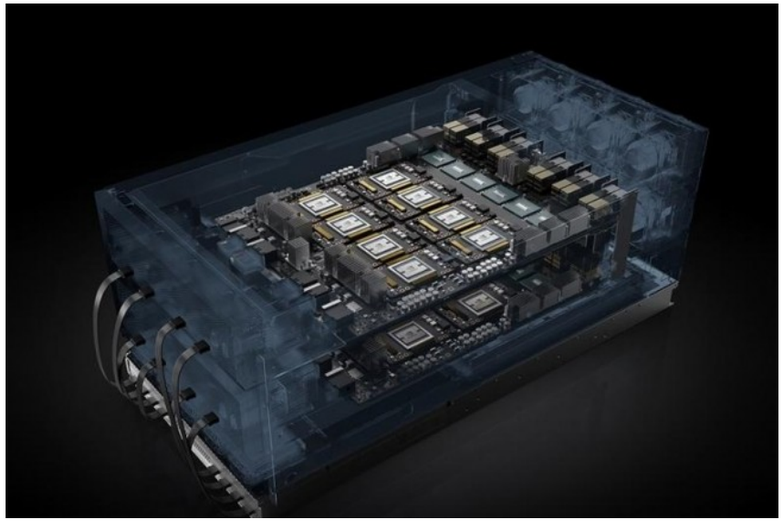
RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

GPU PROGRAMMING

Through external scientific libraries



The complexity of GPUs programming
A complex software environment

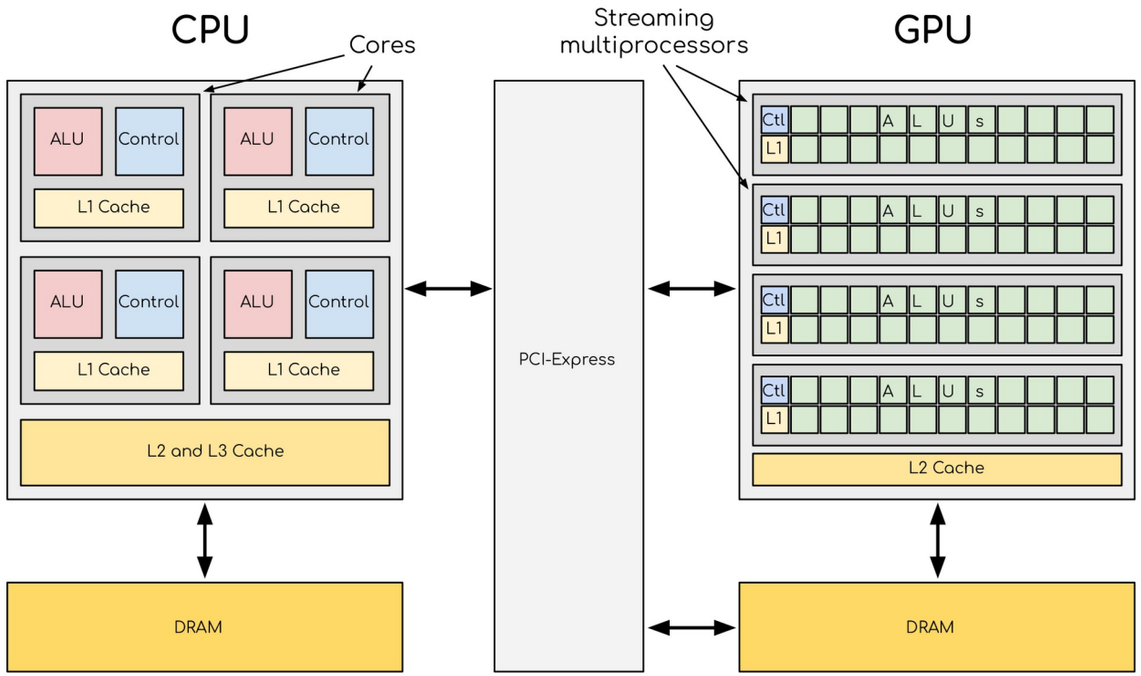
Different hardwares compatibility

Distribution	GCC	Clang	NVHPC	XLC	ArmC/C++	ICC
x86_64	6.x - 13.2	7.x - 17.0	23.x	No	No	2021.7
Arm64 sbsa	6.x - 13.2	7.x - 17.0	22.x	No	23.04.1	No

Compiler vs. Driver/Lib version

Different languages and API

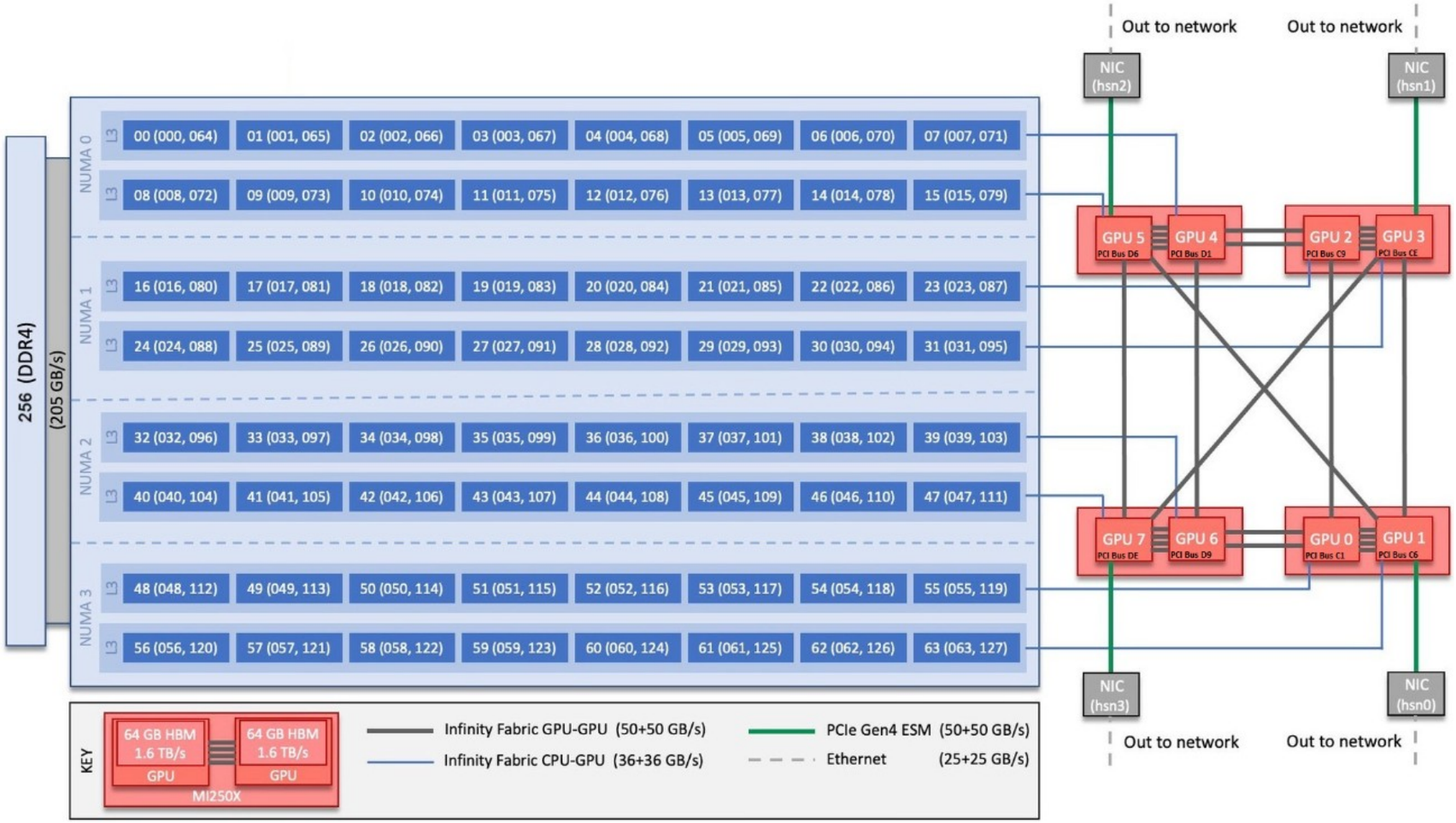
The complexity of GPUs programming
Memory handling




```
int main()
{
    const unsigned int N = 1048576;
    const unsigned int bytes = N * sizeof(int);
    int *h_a = (int*)malloc(bytes);
    int *d_a;
    cudaMalloc((int**)&d_a, bytes);

    memset(h_a, 0, bytes);
    cudaMemcpy(d_a, h_a, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(h_a, d_a, bytes, cudaMemcpyDeviceToHost);

    return 0;
}
```

Data transfers in both directions

The complexity of GPUs programming
Example on Adastra MI250 partition



Complexity of the connections

The complexity of GPUs programming
Example on Adastra MI250 partition

With HSA_ENABLE_SDMA=1

Unidirectional copy peak Gio/s

	CPU NUMA				GPUs							
From/to	0	1	2	3	c1	c6	c9	ce	d1	d6	d9	de
0	-	-	-	-	25.5	25.5	25.5	25.5	25.5	25.5	25.5	25.5
1	-	-	-	-	25.5	25.5	25.5	25.5	25.5	25.5	25.5	25.5
2	-	-	-	-	25.5	25.5	25.5	25.5	25.5	25.5	25.5	25.5
3	-	-	-	-	25.5	25.5	25.5	25.5	25.5	25.5	25.5	25.5
c1	25.2	25.2	25.2	25.2	-	50.9	36.9	37.5	37.5	36.9	51.0	51.0
c6	25.1	25.1	25.1	25.1	51.0	-	37.5	36.9	36.9	36.9	51.0	51.0
c9	25.2	25.2	25.2	25.2	36.9	37.5	-	51.0	51.0	51.0	37.5	36.9
ce	25.2	25.2	25.2	25.2	37.5	36.9	51.0	-	51.0	51.0	36.9	36.9
d1	25.2	25.2	25.2	25.2	37.5	37.0	51.0	51.0	-	51.1	36.9	37.5
d6	25.2	25.2	25.2	25.2	36.9	36.9	51.0	51.0	51.0	-	37.5	36.9
d9	25.2	25.2	25.2	25.2	51.0	51.0	37.5	36.9	36.9	37.5	-	51.0
de	25.2	25.2	25.2	25.2	51.0	51.0	36.9	36.9	37.5	36.9	51.0	-

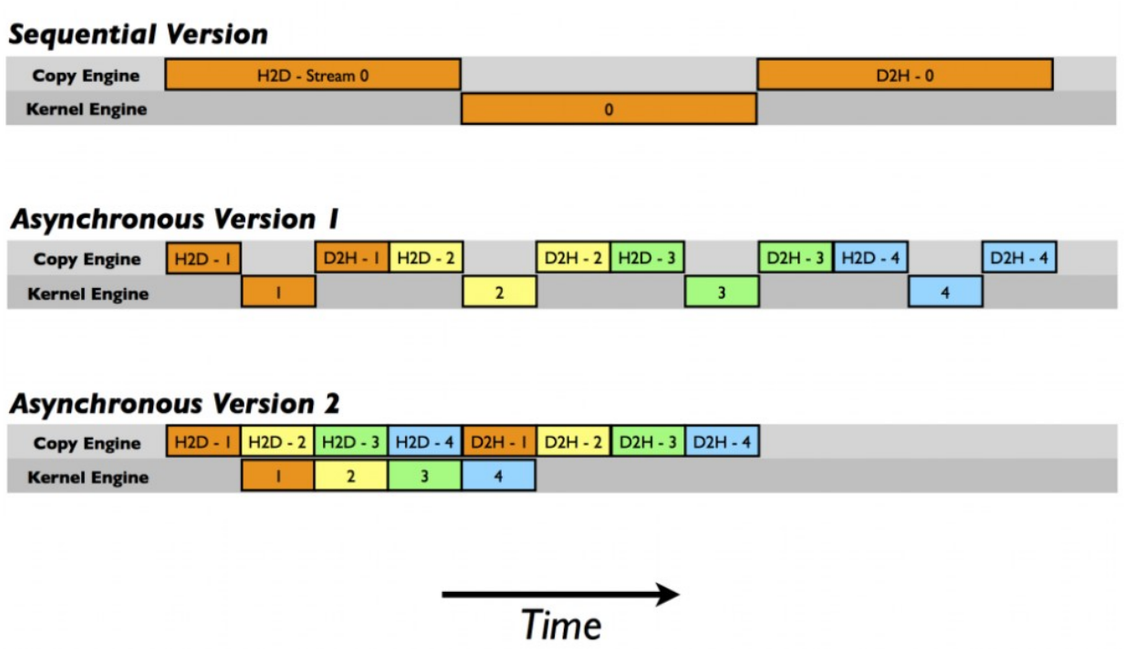
Bidirectional copy peak Gio/s

	CPU NUMA				GPUs							
From/to	0	1	2	3	c1	c6	c9	ce	d1	d6	d9	de
0	-	-	-	-	29.3	29.2	29.3	28.7	29.3	29.2	29.3	28.7
1	-	-	-	-	29.3	29.2	29.3	28.7	29.3	29.3	29.3	28.7
2	-	-	-	-	29.3	29.2	29.3	28.7	29.3	29.2	29.3	28.7
3	-	-	-	-	29.3	29.2	29.3	28.7	29.3	29.2	29.3	28.7
c1	29.3	29.3	29.3	29.3	-	101.5	67.6	69.6	69.6	67.6	101.5	101.5
c6	29.2	29.2	29.2	29.2	101.5	-	69.6	67.6	67.6	67.6	101.9	101.8
c9	29.3	29.3	29.3	29.3	67.6	69.6	-	101.9	101.7	102.0	69.6	67.5
ce	28.7	28.7	28.7	28.7	69.6	67.6	101.9	-	101.3	101.7	67.5	67.7
d1	29.3	29.3	29.3	29.3	69.6	67.6	101.7	101.3	-	101.8	67.6	69.6
d6	29.2	29.3	29.2	29.2	67.6	67.6	102.0	101.7	101.8	-	69.6	67.6
d9	29.3	29.3	29.3	29.3	101.5	101.9	69.6	67.5	67.6	69.6	-	101.5
de	28.7	28.7	28.7	28.7	101.5	101.8	67.5	67.7	69.6	67.6	101.5	-

Data transfer throughput: non uniform

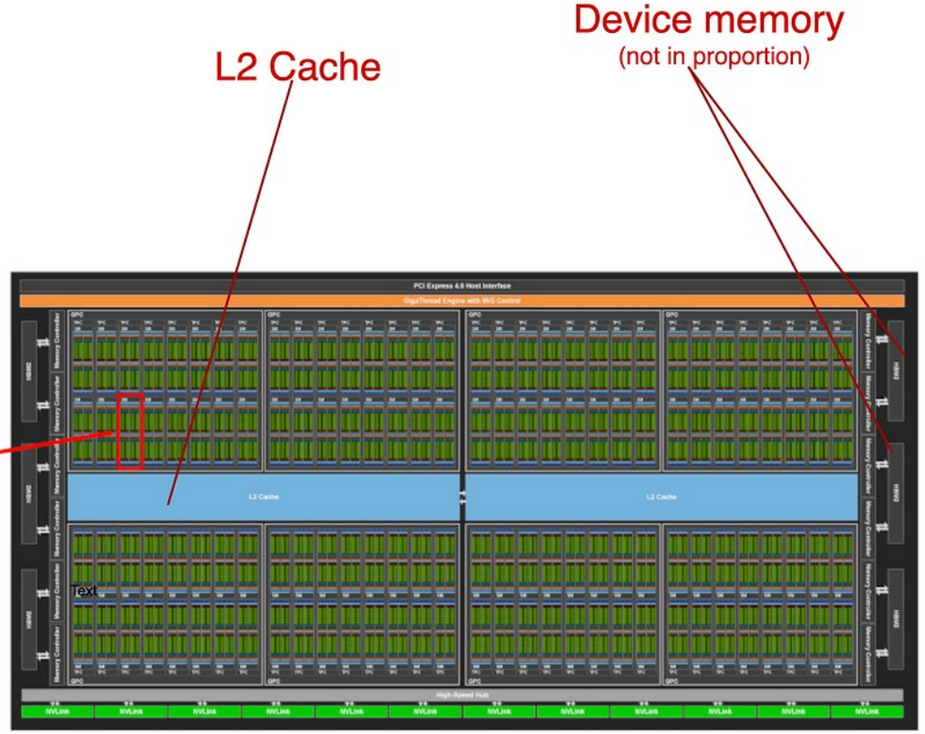
The complexity of GPUs programming
Asynchronism handling

```
for (int i = 0; i < nStreams; ++i) {
  int offset = i * streamSize;
  cudaMemcpyAsync(&d_a[offset], &a[offset], streamBytes, cudaMemcpyHostToDevice, stream[i]);
  kernel<<<streamSize/blockSize, blockSize, 0, stream[i]>>>(d_a, offset);
  cudaMemcpyAsync(&a[offset], &d_a[offset], streamBytes, cudaMemcpyDeviceToHost, stream[i]);
}
```



The complexity of GPUs programming
Fine grain parallelism, high number of parallel tasks

SM



GA100 Full GPU with 128 SMs

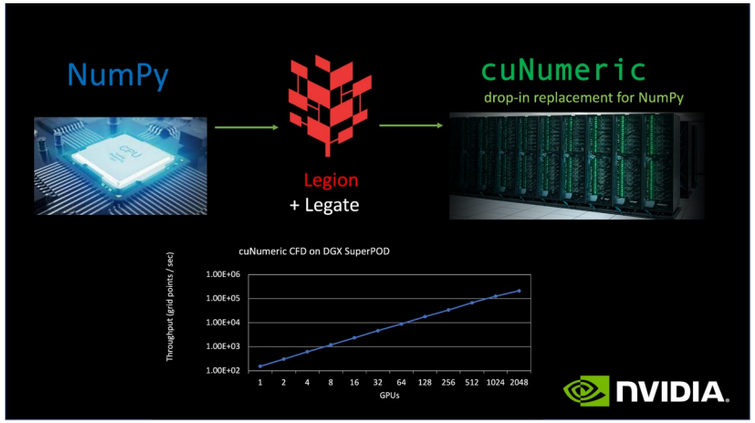
L1 Cache

A100 → ~6k cores, ~700k threads
GEMM N=2000 → 4M parallel tasks

Find the good external library

- Find the nice library with appropriate features
- Avoid white elephant, be precise about features needs
- Software quality: tests, releases, stable API, neat documentation, active issues system
- Allows to focus on our algorithms level

Examples of external libraries
Python ecosystem



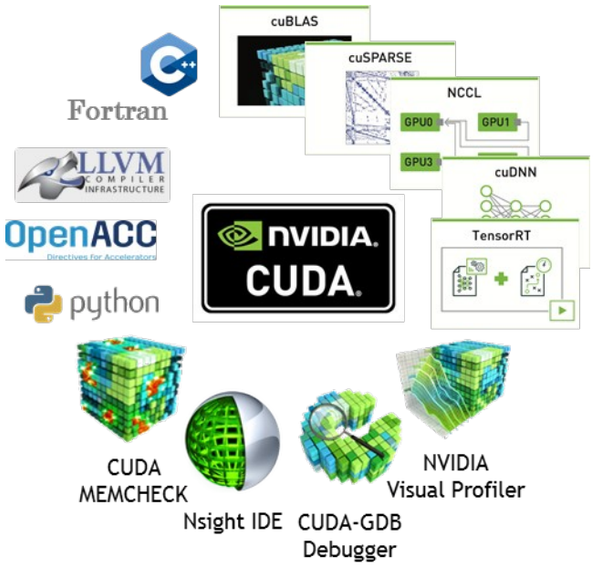
Nvidia, numpy on CUDA GPUs

CuPy

NumPy & SciPy for GPU

Numpy and Scipy on CUDA (+/-AMD Roc) GPUs

Examples of external libraries
C/C++, Fortran ecosystem, GPU vendors libraries



Dense, Sparse and Tensor linear algebra, FFT, DNN
Nvidia GPUs

CUDA Library	ROCm Library	Description
cuBLAS	rocBLAS	Basic Linear Algebra Subroutines
cuFFT	rocFFT	Fast Fourier Transfer Library
cuSPARSE	rocSPARSE	Sparse BLAS + SPMV
cuSolver	rocSolver	Lapack Library
AMG-X	rocALUTION	Sparse iterative solvers & preconditioners with Geometric & Algebraic MultiGrid
Thrust	rocThrust	C++ parallel algorithms library
CUB	rocPRIM	Low Level Optimized Parallel Primitives
cuDNN	MIOpen	Deep learning Solver Library
cuRAND	rocRAND	Random Number Generator Library
EIGEN	EIGEN	C++ template library for linear algebra: matrices, vectors, numerical solvers
NCCL	RCCL	Communications Primitives Library based on the MPI equivalents

AMD ROCm equivalence

Intel oneAPI ...

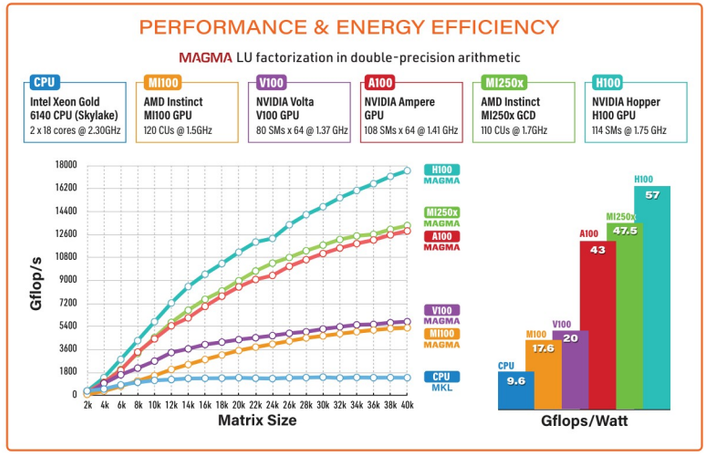
Examples of external libraries

C/C++, Fortran ecosystem, not vendor specific

MAGMA

HYBRID ALGORITHMS

MAGMA uses a hybridization methodology, where algorithms of interest are split into tasks of varying granularity, and their execution is scheduled over the available hardware components. Scheduling can be static or dynamic. In either case, small non-parallelizable tasks, often on the critical path, are scheduled on the CPU, and larger more parallelizable ones, often Level-3 BLAS, are scheduled on the GPU. When CPU-GPU communication overhead becomes significant, the entire computation might run exclusively on the GPU, leading to what is termed as "GPU-native algorithms".

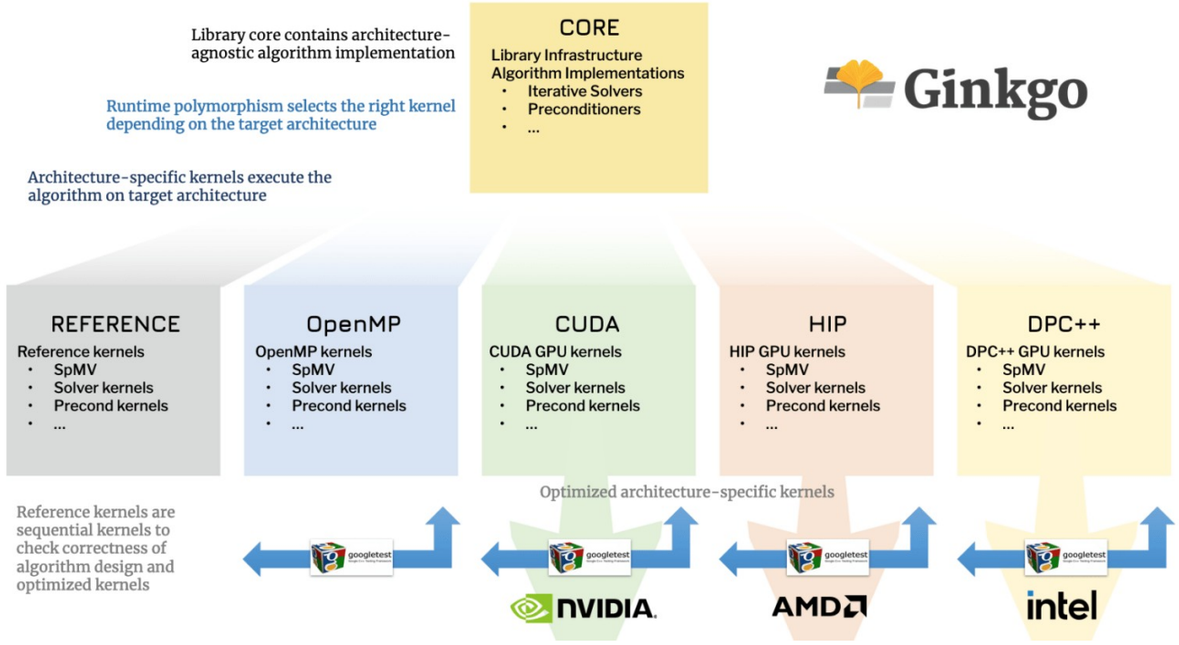


FEATURES AND SUPPORT

- MAGMA 2.7.2 FOR CUDA OR HIP
- MAGMA Pre-release FOR SYCL/DPC++

CUDA	HIP	SYCL Pre-release	
✓	✓	✓	Linear system solvers
✓	✓	✓	Eigenvalue problem solvers
✓	✓	✓	Auxiliary BLAS
✓	✓	✓	Batched LA
✓	✓		Sparse LA
✓	✓	✓	CPU/GPU Interface
✓	✓	✓	Multiple precision support
✓	✓		Mixed precision (including FP16)
✓	✓	✓	Non-GPU-resident factorizations
✓	✓	✓	GPU-only factorizations
✓	✓		Multicore and multi-GPU support
✓	✓		MAGMA DNN v1.4
✓	✓	✓	LAPACK testing
✓	✓	✓	Linux
✓	✓		Windows
✓	✓		macOS

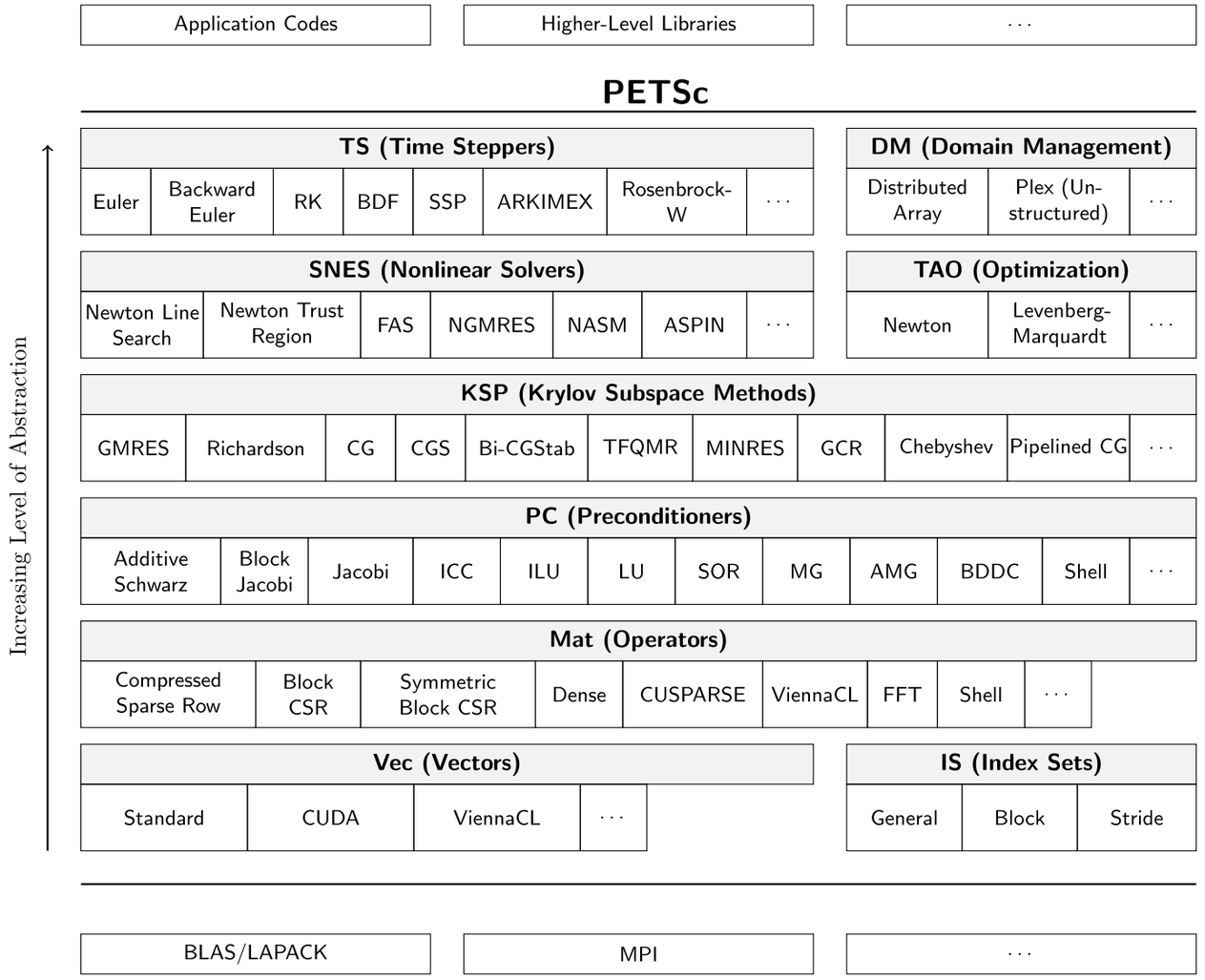
Ginkgo

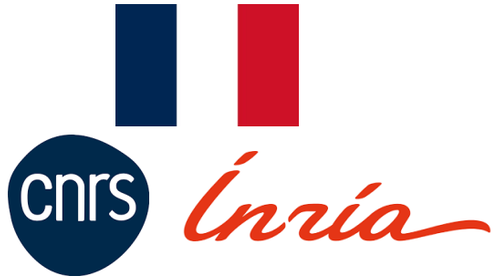


Linear algebra, 1 node of CPUs + GPUs

Examples of external libraries

C/C++, Fortran ecosystem, Distributed (MPI)

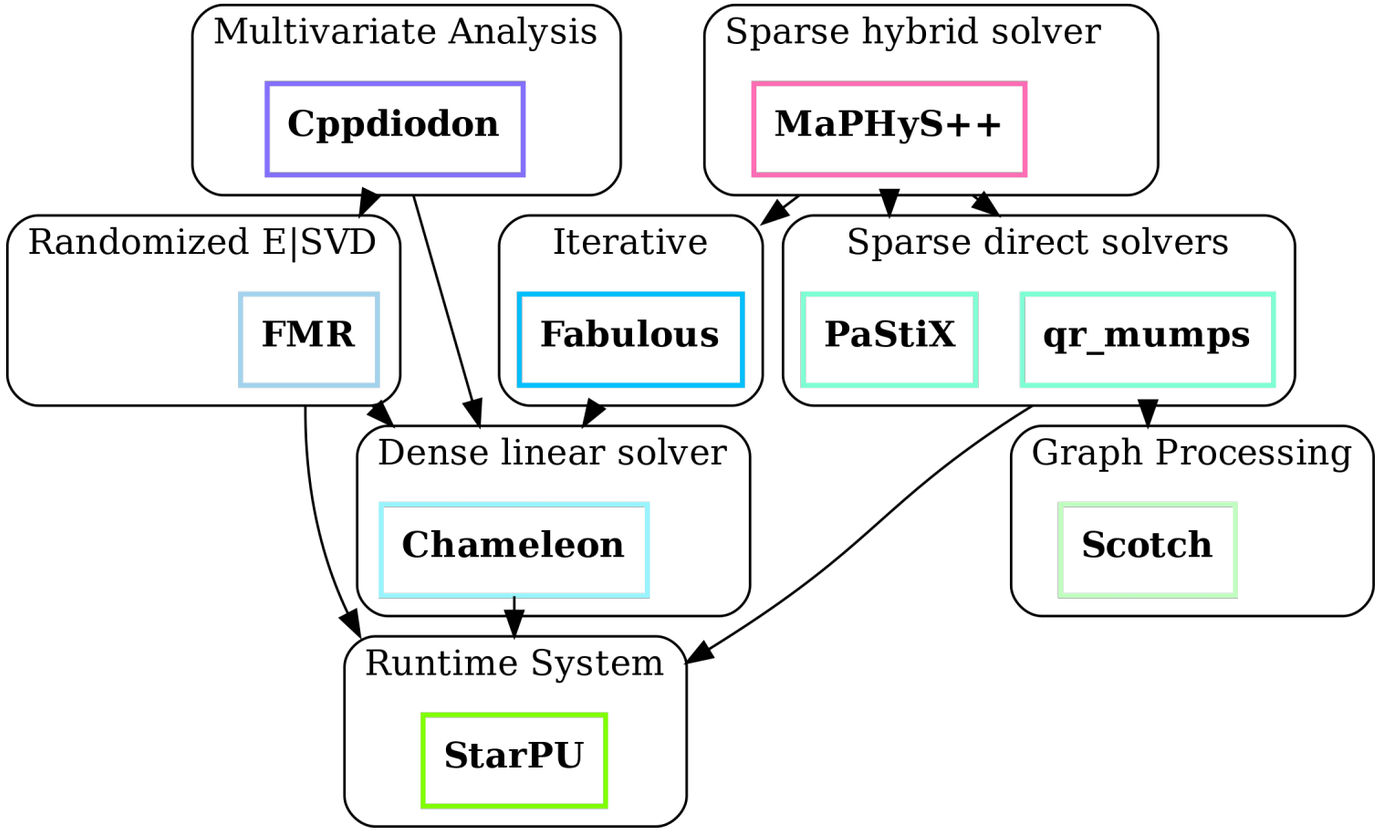




- MUMPS
- PaStiX
- HPDDM

Examples of external libraries
C/C++, Fortran ecosystem, using StarPU runtime system

https://solverstack.gitlabpages.inria.fr/



Dense and Sparse linear algebra, multi nodes CPUs + GPUs

GEMM directly with CUDA

```cpp
int main(){
    // Perform matrix multiplication C = A*B
    // where A, B and C are NxN matrices
    int N = 16;
    int SIZE = N*N;

    // Allocate memory on the host
    vector<float> h_A(SIZE);
    vector<float> h_B(SIZE);
    vector<float> h_C(SIZE);

    // Initialize matrices on the host
    for (int i=0; i<N; i++){
        for (int j=0; j<N; j++){
            h_A[i*N+j] = sin(i);
            h_B[i*N+j] = cos(j);
        }
    }

    // Allocate memory on the device
    dev_array<float> d_A(SIZE);
    dev_array<float> d_B(SIZE);
    dev_array<float> d_C(SIZE);

    // Copy data host to device
    d_A.set(&h_A[0], SIZE);
    d_B.set(&h_B[0], SIZE);

    // Compute multiplication
    matrixMultiplication(d_A.getData(), d_B.getData(), d_C.getData(), N);
    cudaDeviceSynchronize();

    // Copy data back to host from device
    d_C.get(&h_C[0], SIZE);
    cudaDeviceSynchronize();
```

GEMM CUDA

- Need to handle host and device arrays
- Need to code a kernel
- Need to handle each GPU one by one
- Need to handle asynchronism

Using external libraries : example Chameleon

```cpp
#include <chameleon.h>
#include <vector>
using namespace std;
int main() {
    // Perform matrix multiplication C = A*B
    // where A, B and C are NxN matrices
    int N = 16;
    int SIZE = N*N;

    // Allocate memory on the host
    vector<float> A(SIZE);
    vector<float> B(SIZE);
    vector<float> C(SIZE);

    // Initialize matrices on the host
    for (int i=0; i<N; i++){
        for (int j=0; j<N; j++){
            A[i*N+j] = sin(i);
            B[i*N+j] = cos(j);
        }
    }

    CHAMELEON_Init( 32, 4 );
    CHAMELEON_sgemm( ChamNoTrans, ChamNoTrans, N, N, N,
                     1.0, A.data(), N,
                          B.data(), N,
                     0.0, C.data(), N );
    CHAMELEON_Finalize();
```

GEMM Chameleon (StarPU)

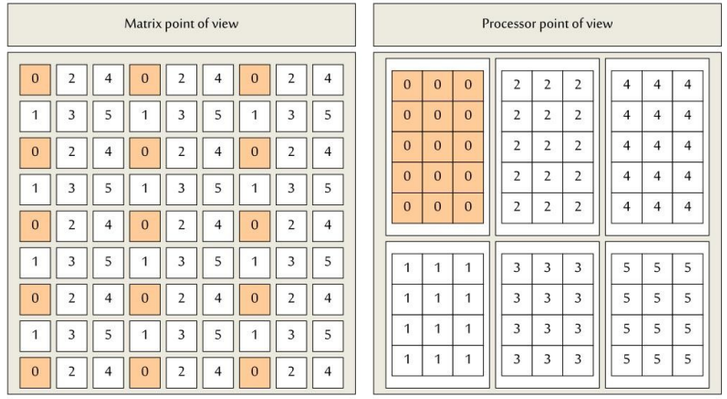
- Stay focused on traditionnal host arrays
- No need to code the function, just call it,
  (a data conversion is often necessary)
- Execute on any number of workers (CPUs + GPUs)
- Can call several algorithms without synchronization
  to let tasks being interleaved = optimization

Using external libraries, Distributed (MPI)

```c
int main() {
    // Perform matrix multiplication C = A*B
    // where A, B and C are NxN matrices
    int N = 16;
    int NB = 2;
    int SIZE = N*N;

    CHAMELEON_Init( 32, 4 );
    CHAMELEON_Set( CHAMELEON_TILE_SIZE, NB );
    int NMPI = CHAMELEON_Comm_size();
    int GRID_P = NMPI;
    int GRID_Q = 1;

    /* Initialize the structure required for CHAMELEON tile interface */
    CHAM_desc_t *descA = NULL, *descB = NULL, *descC = NULL;
    CHAMELEON_Desc_Create(&descA, NULL, ChamRealFloat,
        NB, NB, NB*NB, N, N, 0, 0, N, N, GRID_P, GRID_Q);
    CHAMELEON_Desc_Create(&descB, NULL, ChamRealFloat,
        NB, NB, NB*NB, N, N, 0, 0, N, N, GRID_P, GRID_Q);
    CHAMELEON_Desc_Create(&descC, NULL, ChamRealFloat,
        NB, NB, NB*NB, N, N, 0, 0, N, N, GRID_P, GRID_Q);

    struct data_fill fill_args_A = { 1.0 };
    CHAMELEON_map_Tile(ChamW, ChamUpperLower, descA, fill_cpu, &fill_args_A);

    struct data_fill fill_args_B = { 2.0 };
    CHAMELEON_map_Tile(ChamW, ChamUpperLower, descB, fill_cpu, &fill_args_B);

    CHAMELEON_sgemm_Tile(ChamNoTrans, ChamNoTrans, 1.0, descA, descB, 0.0, descC);

    CHAMELEON_Finalize();
```

MPI GEMM Chameleon (StarPU)

2D Block Cyclic Layout



- User can provide a function to fill a given tile, with its specific metadata (some parameters, a file, ...)

```c
struct data_fill {
  float value;
};

static int fill_cpu(const CHAM_desc_t *desc,
    cham_uplo_t uplo, int m, int n,
    CHAM_tile_t *tile, void *user_data)
{
    struct data_fill *data = (struct data_fill *)user_data;
    float *T = (float *)tile->mat;

    /* Get the dimension of the tile */
    int tempmm = (m == (desc->mt-1)) ? (desc->m - m * desc->mb) : desc->mb;
    int tempnn = (n == (desc->nt-1)) ? (desc->n - n * desc->nb) : desc->nb;

    /* fill the tile with the user's value */
    for (int i=0; i<tempmm*tempnn; i++){
        T[i] = data->value;
    }

    (void)uplo;
    return 0;
}
```

Merci. Des questions ?