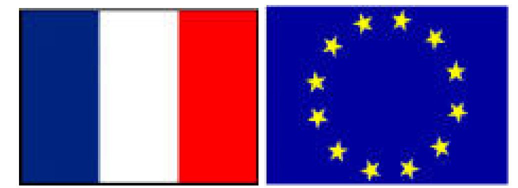
POLYTECH
PARIS-SACLAY



GP-GPU

TD2: CUBLAS library

Stéphane Vialle

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

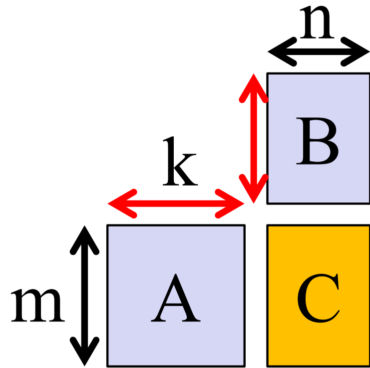
Stephane.Vialle@centralesupelec.fr
http://www.metz.supelec.fr/~vialle

CUBLAS dgemm / sgemm

API :

```
cublasStatus_t cublasDgemm(cublasHandle_t handle,
                           cublasOperation_t transa,
                           cublasOperation_t transb,
                           int m, int n, int k,
                           const double *alpha,
                           const double *A, int lda,
                           const double *B, int ldb,
                           const double *beta,
                           double *C, int ldc)
```

$$C = \alpha . op(A) \times op(B) + \beta . C$$



- We multiply square matrices of SIZE × SIZE éléments
- Stored in SIZE × SIZE memory arrays

→ m, n, k, lda, ldb, ldc ?

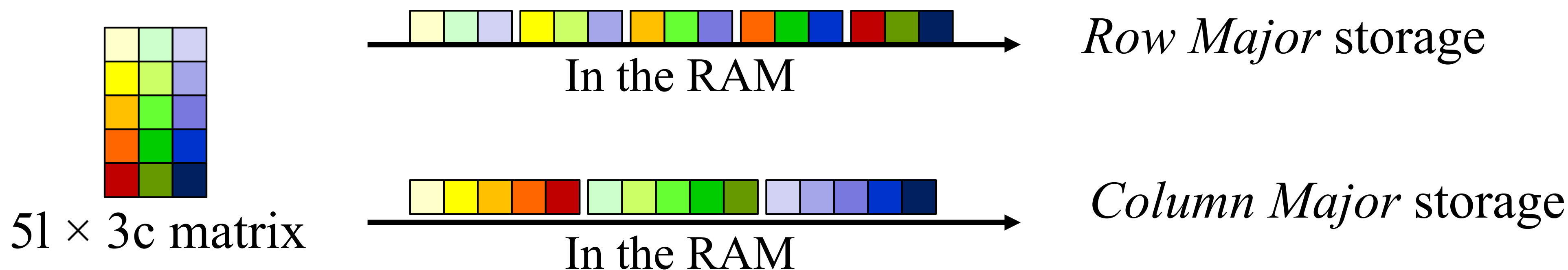
Do we need to transpose ?

We need the pointers on the GPU matrices:
→ the addresses of the symbols GPU_A, GPU_B, GPUC

POLYTECH PARIS-SACLAY

CUBLAS dgemm / sgemm

Row Major / Column Major data storage:



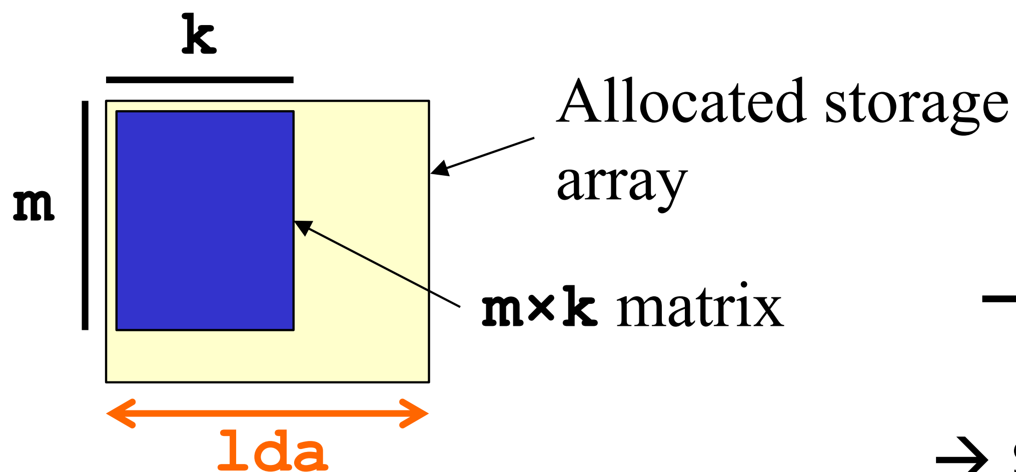
- **`cblas`** on CPU: you can choose the storage
- **`cublas`** on GPU: *Column Major* is assumed

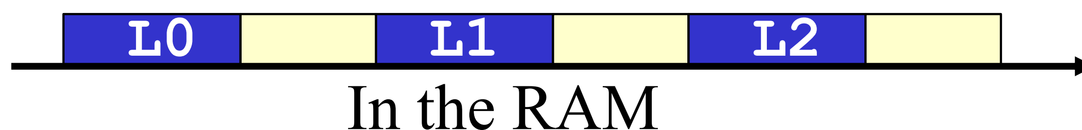
  But you can specify to transpose the operands

  … but result will be stored in Column Major

CUBLAS dgemm / sgemm

With *Row Major* data storage (line by line) on CPU:





→ Skip **`lda-k`** elements at the end of each line before to access the next line

→ Library needs to know **`k`** and **`lda`**

For *Column Major* storage : consider vertical **`lda`** and **`m`** ...

… and inverse these rules if you store transposed matrices on GPU

CUBLAS dgemm / sgemm

More complex on GPU !

A	device	input	<type> array of dimensions `lda x k` with `lda>=max(1,m)` if `transa == CUBLAS_OP_N` and `lda x m` with `lda>=max(1,k)` otherwise.
lda		input	leading dimension of two-dimensional array used to store the matrix `A`.
B	device	input	<type> array of dimension `ldb x n` with `ldb>=max(1,k)` if `transa == CUBLAS_OP_N` and `ldb x k` with `ldb>=max(1,n)` otherwise.
ldb		input	leading dimension of two-dimensional array used to store matrix `B`.

But in our very simple case:

```
T_real GPU_A[SIZE][SIZE];
        m = n = k = SIZE
        lda = ldb = ldc = SIZE
```

CUBLAS dgemm / sgemm

GPU/CUDA symbols are not variables

```
__device__ T_real GPU_A[SIZE][SIZE]; // Symbol
T_real *AdrGPU_A = NULL;             // Pointer
```

You can not use the '&' operator to get the address of a symbol:

```
AdrGPU_A = ~~&GPU_A[0][0];~~
```

You have to use:

```
cudaGetSymbolAddress( (void **)&AdrGPU_A, GPU_A )
```

Returns: **cudaSuccess** or an **error code**

→ Use: **CHECK_CUDA_SUCCESS(…)** during the lab

Extract address and set a pointer on each matrix:

GPU_A, GPU_B, GPU_C …

CUBLAS dgemm / sgemm – Q1

1 - Implement: GPU_C = GPU_A × GPU_B

→ cublasSgemm(…)

Do you need to transpose some matrices ?

CUBLAS dgemm / sgemm – Q2.1

2.1 – Interest of matrix transposition kernel v1 ?

```
#define BLOCK_SIZE_XY_TRANSPO ......
// Without Shared Memory
__global__ void TransposeKernel_v0(
      T_real *MT, T_real *M, int mLig, int nCol)
// With Shared Memory
__global__ void TransposeKernel_v1(
      T_real *MT, T_real *M, int mLig, int nCol)
```

CUBLAS dgeam / sgeam – Q2.2

Transpose a matrix with dgeam / sgeam

Compute: $C = \alpha.op(A) + \beta.op(B)$

```
cublasStatus_t cublasDgeam(
        cublasHandle_t handle,
        cublasOperation_t transa,
        cublasOperation_t transb,
        int m, int n,
        const double *alpha,
        const double *A, int lda,
        const double *beta,
        const double *B, int ldb,
        double *C, int ldc)
```

`cublasDgeam` is not included in `cblas` library/standard.

→ it is a `cublas` extension of the `cblas` standard

Note: **NULL** pointer can be used for an unused matrix

$$op(A) = \begin{cases} A & \text{if transa == CUBLAS_OP_N} \\ A^T & \text{if transa == CUBLAS_OP_T} \\ A^H & \text{if transa == CUBLAS_OP_C} \end{cases}$$

POLYTECH PARIS-SACLAY

CUBLAS dgeam / sgeam – Q2.2

2.2 - Transpose a GPU_C matrix with **`cublasDgeam`**

CUBLAS dgemm / sgemm – Q3

Optimal solution using only dgemm/sgemm:

- Using `CUBLAS_OP_T` or `CUBLAS_OP_N` in `cublasDgemm`
  → You can transpose or not transpose the operands

- And matrix product transposition has some interesting properties…

3 - Implement a complete matrix product $C = A \times B$

- With `cublasSgemm` only
- And with matrices in *Row Major* storage (classical C/C++ storage)

CUBLAS cublasGemmEx – Q4

Using Tensor Cores with `cublasGemmEx`:

- Using `CUBLAS_OP_T` or `CUBLAS_OP_N` in `cublasGemmEx`

  → You can transpose or not transpose the operands
- You can specify A, B, C matrix datatypes
- You can specify compute types

4.1 - Implement a complete matrix product C = A × B in 32bits

- With `cublasGemmEx` only
- With matrices in *Row Major* storage (classical C/C++ storage)
- With 32bits datatype CUDA_R_32F
- With 32bits compute type CUBLAS_COMPUTE_32F

POLYTECH PARIS-SACLAY

TD 2

CUBLAS cublasGemmEx – Q4

Using Tensor Cores with `cublasGemmEx`:

- Using `CUBLAS_OP_T` or `CUBLAS_OP_N` in `cublasGemmEx`

  → You can transpose or not transpose the operands
- You can specify A, B, C matrix datatypes
- You can specify compute types

4.2 - Implement a complete matrix product C = A × B in 32/16 bits

- With `cublasGemmEx` only
- With matrices in *Row Major* storage (classical C/C++ storage)
- With 32bits datatype CUDA_R_32F
- With compute types :
  - CUBLAS_COMPUTE_32F_FAST_TF32
  - CUBLAS_COMPUTE_32F_FAST_16F
  - CUBLAS_COMPUTE_32F_FAST_16BF

TD2: CUBLAS library

Fin