

CentraleSupélec  

Mineure CalHau2

## CUBLAS library

**Stéphane Vialle**

 universit  Paris-Saclay   Sciences et technologies de l'information et de la communication (STIC)

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

1

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

## CUBLAS library

- 1 – Les biblioth ques BLAS
- 2 – CUBLAS vs BLAS
- 3 – La fonction *cublasDgemv*
- 4 – Transposition de matrice
- 5 – Espace de stockage pour *cublasDgemv*
- 6 – CUBLAS avec TensorCores
- 7 – Adresse m moire d'un *symbol*

2

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

Biblioth ques BLAS et CUBLAS

## Les biblioth ques « BLAS »

**BLAS : Basic Linear Algebra Subprograms**

- Ensemble de fonctions d'alg bre lin aire
- Prototypes publi s en 1979  
[https://fr.wikipedia.org/wiki/Basic\\_Linear\\_Algebra\\_Subprograms](https://fr.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms)
- 3 niveaux de BLAS :
  - niveau 1 : op rations sur les vecteurs
  - niveau 2 : op rations de type matrice – vecteur
  - niveau 3 : op rations de type matrice – matrice
- UNE API standardis e et DES implantations
  - implantations open-sources  
ex : OpenBLAS, ATLAS (tr s efficaces)
  - implantations propri taires cibl es sur un type de mat riel  
ex : biblioth que MKL d'Intel
  - installer en recompilant sur la machine cible

3

---

---

---

---

---

---

---

---

---

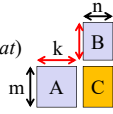
---

Bibliothèques BLAS et CUBLAS

## Les bibliothèques « BLAS »

**La Fonction `cblas_dgemm` :**

- Fonction de produit de matrices denses
- Travaille sur des *double* (`cblas_sgemm` pour les *float*)

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


```

void cblas_dgemm (
  const CBLAS_LAYOUT Layout,
  const CBLAS_TRANSPOSE transa,
  const CBLAS_TRANSPOSE transb,
  const int m, const int n, const int k,
  const double alpha,
  const double *a, const int lda,
  const double *b, const int ldb,
  const double beta,
  double *c, const int ldc)

```

Stockage en row major ou column major

4

---

---

---

---

---

---

---

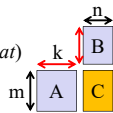
---

Bibliothèques BLAS et CUBLAS

## Les bibliothèques « BLAS »

**La Fonction `cblas_dgemm` :**

- Fonction de produit de matrices denses
- Travaille sur des *double* (`cblas_sgemm` pour les *float*)

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


```

void cblas_dgemm (
  const CBLAS_LAYOUT Layout,
  const CBLAS_TRANSPOSE transa,
  const CBLAS_TRANSPOSE transb,
  const int m, const int n, const int k,
  const double alpha,
  const double *a, const int lda,
  const double *b, const int ldb,
  const double beta,
  double *c, const int ldc)

```

Transposition « au vol » (ou pas) de A et B

5

---

---

---

---

---

---

---

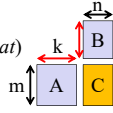
---

Bibliothèques BLAS et CUBLAS

## Les bibliothèques « BLAS »

**La Fonction `cblas_dgemm` :**

- Fonction de produit de matrices denses
- Travaille sur des *double* (`cblas_sgemm` pour les *float*)

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


```

void cblas_dgemm (
  const CBLAS_LAYOUT Layout,
  const CBLAS_TRANSPOSE transa,
  const CBLAS_TRANSPOSE transb,
  const int m, const int n, const int k,
  const double alpha,
  const double *a, const int lda,
  const double *b, const int ldb,
  const double beta,
  double *c, const int ldc)

```

Tailles des matrices

6

---

---

---

---

---

---

---

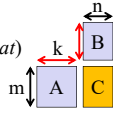
---

Bibliothèques BLAS et CUBLAS

## Les bibliothèques « BLAS »

**La Fonction `cblas_dgemm` :**

- Fonction de produit de matrices denses
- Travaille sur des *double* (`cblas_sgemm` pour les *float*)

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


```

void cblas_dgemm (
  const CBLAS_LAYOUT Layout,
  const CBLAS_TRANSPOSE transa,
  const CBLAS_TRANSPOSE transb,
  const int m, const int n, const int k,
  const double alpha,
  const double *a, const int lda,
  const double *b, const int ldb,
  const double beta,
  double *c, const int ldc)

```

lda, ldb, ldc : tailles des zones de stockage (voir plus loin)

7

---

---

---

---

---

---

---

---

---

---

## CUBLAS library

- 1 – Les bibliothèques BLAS
- 2 – CUBLAS vs BLAS
- 3 – La fonction `cublasDgemm`
- 4 – Transposition de matrice
- 5 – Espace de stockage pour `cublasDgemm`
- 6 – CUBLAS avec TensorCores
- 7 – Adresse mémoire d'un *symbol*

8

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

## 2 – CUBLAS vs BLAS

- Mêmes fonctionnalités d'algèbre linéaire que les BLAS
- Mais quelques différences :
  - Besoin d'initialiser l'usage de la bibliothèque**
    - Récupération d'un « *handle* » sur la bibliothèque
    - Passage de ce *handle* en paramètre de toutes les fonctions CUBLAS
  - Le format « *column major* » (style FORTRAN) est imposé**
    - Possibilité de convertir les données d'entrée au vol
    - Besoin de convertir les données de sortie
  - La bibliothèque ne se charge pas des transferts CPU/GPU**
    - les transferts restent à la charge du développeur
  - Quelques nouvelles fonctions disponibles**
    - des « extensions des BLAS » : non standard mais pratiques !

9

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS  
2 – CUBLAS vs BLAS

**Utilisation des CUBLAS**

1. Ajout de `#include <cublas_v2.h>` au début du fichier src Cuda
2. Ajout de `-lcublas` à l'édition de lien
3. Initialisation des CUBLAS dans le code Cuda
 

```
cublasHandle_t handle;
cublasCreate(&handle); // Return a cudaStatus_t
                        // CUBLAS_STATUS_SUCCESS ?
cublasXxxx(handle,...); // Cublas usage
cublasDestroy(handle); // End of cublas usage
```

Rmq : L'initialisation (`cublasCreate`) peut prendre un peu de temps!  
Ne la faire qu'une seule fois au début du pgm.

10

---

---

---

---

---

---

---

---

CUBLAS library

- 1 – Les bibliothèques BLAS
- 2 – CUBLAS vs BLAS
- 3 – La fonction *cublasDgemv***
- 4 – Transposition de matrice
- 5 – Espace de stockage pour *cublasDgemv*
- 6 – CUBLAS avec TensorCores
- 7 – Adresse mémoire d'un *symbol*

11

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS  
3 – La fonction *cublasDgemv*

**API :**

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

```
cublasStatus_t cublasDgemv(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)
```

**Comparé aux BLAS :**

- On passe le « *handle* » des cublas (1<sup>er</sup> paramètre)
- Les valeurs « *alpha* » et « *beta* » ne sont pas des constantes mais des adresse de variables (constantes)
- On ne précise pas le format des données :  
→ le stockage « *column major* » est imposé

12

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

### 3 – La fonction *cublasSgemv*

API :

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

```

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

```

**Comparé aux BLAS :**

- On passe le « *handle* » des cublas (1<sup>er</sup> paramètre)
- Les valeurs « *alpha* » et « *beta* » ne sont pas des constantes mais des adresse de variables (constantes)
- On ne précise pas le format des données :  
→ le stockage « *column major* » est imposé

13

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

### 3 – La fonction *cublasDgemv*

**Format des données (matrices) :**

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

```

cublasDgemv(
cublasHandle_t handle,
cublasOperation_t transa,
cublasOperation_t transb,
int m, int n, int k,.....)

```

A est une matrice :

- mathématiquement de *m lignes* × *k colonnes*,
- considérée stockée au format *column major* (imposé),
- que l'on peut transposer « au vol » :

$$op(A) = \begin{cases} A & \text{if transa} == \text{CUBLAS\_OP\_N} \\ A^T & \text{if transa} == \text{CUBLAS\_OP\_T} \\ A^H & \text{if transa} == \text{CUBLAS\_OP\_C} \end{cases} \rightarrow \text{matrice conjuguée}$$

Idem pour B : matrice de *k lignes* × *n colonnes*

14

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

### 3 – La fonction *cublasDgemv*

**Format des données (matrices) :**

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

```

cublasDgemv(
cublasHandle_t handle,
cublasOperation_t transa,
cublasOperation_t transb,
int m, int n, int k,.....)

```

A est une matrice :

- mathématiquement de *m lignes* × *k colonnes*,
- considérée stockée au format *column major* (imposé),
- que l'on peut transposer « au vol » :

**En langage C/C++ (comme en CUDA) un tableau 2D est stocké en *row major*...**

...en demandant la transposition au vol d'une matrice (avec `transa = CUBLAS_OP_T`) on l'obtient en *column major* !

15

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

## 3 – La fonction *cublasDgemm*

**Format des données (matrices) :**

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

```
cublasDgemm(
  cublasHandle_t handle,
  cublasOperation_t transa,
  cublasOperation_t transb,
  int m, int n, int k,.....)
```

**Mais** la matrice résultat (C) :

- est générée au format *column major* (imposé)
- donc on obtient  $C^T$  au format naturel du langage C/C++/CUDA

**Pour obtenir C au format du C/C++/CUDA on peut :**

**Sol 1 :** Transposer C avec un kernel écrit pour l'occasion

**Sol 2 :** Transposer C avec la fonction CUBLAS `cublasDgeam` (qui n'est pas dans les BLAS mais dans CUBLAS)

**Sol 3 :** Organiser les calculs matriciels pour calculer  $C^T$  et donc obtenir C dans le format du C/C++/CUDA

16

---

---

---

---

---

---

---

---

---

---

CUBLAS library

- 1 – Les bibliothèques BLAS
- 2 – CUBLAS vs BLAS
- 3 – La fonction *cublasDgemm*
- 4 – Transposition de matrice**
- 5 – Espace de stockage pour *cublasDgemm*
- 6 – CUBLAS avec TensorCores
- 7 – Adresse mémoire d'un *symbol*

17

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

## 4 – Transposition de matrice

**Sol 1 : kernel de transposition**

```
__global__ void TransposeKernel_v0(float *MT, float *M,
                                  int mLig, int nCol)
{
  int lig = threadIdx.y + blockIdx.y*BSIZE_XY_KT0;
  int col = threadIdx.x + blockIdx.x*BSIZE_XY_KT0;
  if (lig < mLig && col < nCol)
    MT[col*mLig + lig] = M[lig*nCol + col];
}
```

Accès NON coalescent

Accès coalescent

→ Utiliser la *shared memory* pour être coalescent à la lecture ET à l'écriture...

18

---

---

---

---

---

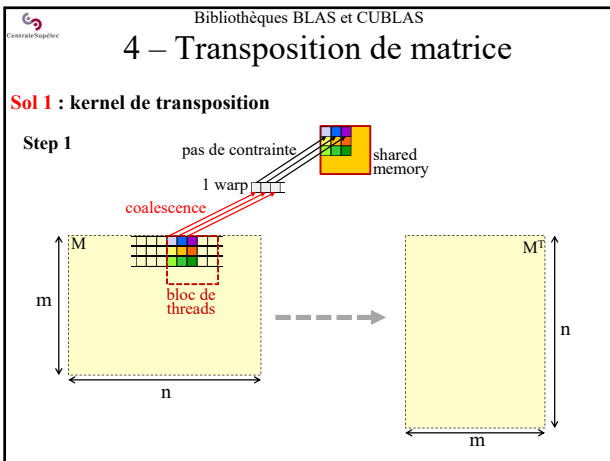
---

---

---

---

---



19

---

---

---

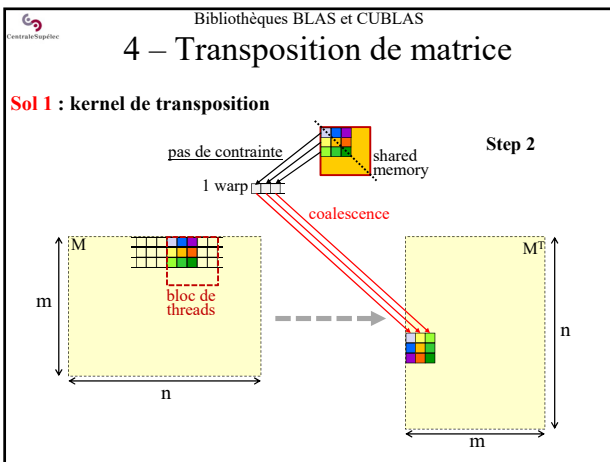
---

---

---

---

---



20

---

---

---

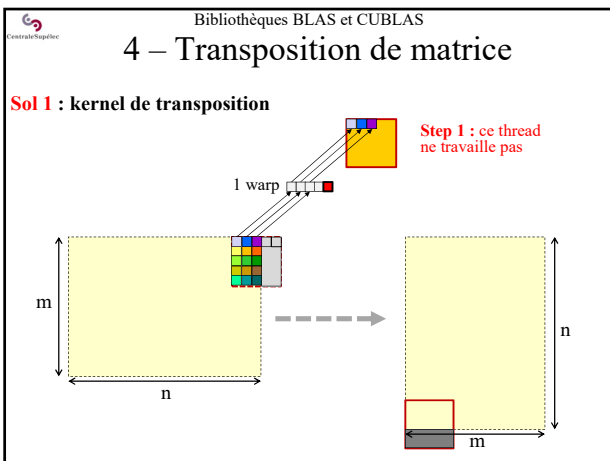
---

---

---

---

---



21

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

## 4 – Transposition de matrice

**Sol 1 : kernel de transposition**

1 warp

Step 1 : ce thread ne travaille pas

Step 2 : ce thread travaille !

m

n

→ Deux conditions différentes par thread  
→ Deux « boundaries » par threads

A revoir dans le cours sur la *shared memory*

22

---

---

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

## 4 – Transposition de matrice

**Sol 1 : kernel de transposition**

```

global__ void TransposeKernel_v1(float *MT, float *M,
                                int mLig, int nCol)
{
int firstLibBlock = blockIdx.y*BSIZE_XY_KT1;
int firstColBlock = blockIdx.x*BSIZE_XY_KT1;
int lig = firstLibBlock + threadIdx.y;
int col = firstColBlock + threadIdx.x;
int ligT = firstColBlock + threadIdx.y;
int colT = firstLibBlock + threadIdx.x;

__shared__ float shM[BSIZE_XY_KT1][BSIZE_XY_KT1];
if (lig < mLig && col < nCol) // Condition 1
shM[threadIdx.y][threadIdx.x] = M[lig*nCol + col];
__syncthreads();
if (ligT < nCol && colT < mLig) // Condition 2
MT[ligT*mLig + colT] = shM[threadIdx.x][threadIdx.y];
}

```

23

---

---

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

## 4 – Transposition de matrice

**Sol 2 : utilisation de cublasDgeam :**

Réalise le calcul de :  $C = \alpha.op(A) + \beta.op(B)$

```

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)

```

Voir aussi `cublasSgeam(...)`

→ Permet le calcul (très rapide) d'une transposée de matrice

Rmq : on peut passer un ptr NULL si on ne sert pas d'une matrice et qu'on ne fait pas d'opération dessus

→ TD/TP 2

24

---

---

---

---

---

---

---

---

---

---

---

---



Bibliothèques BLAS et CUBLAS

## 4 – Transposition de matrice

**Sol 3 : calcul de  $C^T$  en stockage *column major* :**

A partir de A et B, calculer  $C^T$  au lieu de C ( $=A \times B$ )  
 ... grâce aux propriétés de l'algèbre linéaire et des CUBLAS...

→ Donc obtenir  $C^T$  stockée en *column major*  
 → Donc obtenir C dans un tableau C/C++/CUDA en *row major*

TD/TP 2

25

---

---

---

---

---

---

---

---

CUBLAS library

- 1 – Les bibliothèques BLAS
- 2 – CUBLAS vs BLAS
- 3 – La fonction *cublasDgemv*
- 4 – Transposition de matrice
- 5 – Espace de stockage pour *cublasDgemv***
- 6 – CUBLAS avec TensorCores
- 7 – Adresse mémoire d'un *symbol*

26

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

## 5 – Espace de stockage pour *cublasDgemv*

**Autres paramètres (doc NVIDIA)**

| Param. | Memory         | In/out | Meaning   |
|--------|----------------|--------|---|
| handle |                | input  | handle to the CUBLAS library context.   |
| transa |                | input  | operation op(A) that is non- or (conj.) transpose.  |
| transb |                | input  | operation op(B) that is non- or (conj.) transpose.  |
| m      |                | input  | number of rows of matrix op(A) and c.   |
| n      |                | input  | number of columns of matrix op(A) and c.  |
| k      |                | input  | number of columns of op(A) and rows of op(B).   |
| alpha  | host or device | input  | <type>-scalar used for multiplication.  |
| A      | device         | input  | <type>-array of dimensions $l_{da} \times k$ with $l_{da} = \max(1, m)$ if <i>transa</i> == CUBLAS_OP_N and $l_{da} \times m$ with $l_{da} = \max(1, k)$ otherwise. |
| lda    |                | input  | leading dimension of two-dimensional array used to store the matrix A.  |
| B      | device         | input  | <type>-array of dimension $l_{db} \times n$ with $l_{db} = \max(1, k)$ if <i>transa</i> == CUBLAS_OP_N and $l_{db} \times n$ with $l_{db} = \max(1, n)$ otherwise.  |
| ldb    |                | input  | leading dimension of two-dimensional array used to store matrix B.  |
| beta   | host or device | input  | <type>-scalar used for multiplication. If <i>beta</i> =0, C does not have to be a valid input.  |
| C      | device         | in/out | <type>-array of dimensions $l_{dc} \times n$ with $l_{dc} = \max(1, m)$ .   |
| ldc    |                | input  | leading dimension of a two-dimensional array used to store the matrix C.  |

Les tableaux de stockages de A, B et C, peuvent être plus grands que les matrices

On doit préciser leur dimension principale...

27

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

## 5 – Espace de stockage pour *cublasDgemv*

Les espaces de stockage peuvent être plus grand que les matrices :

- Réutilisation d'un espace alloué précédemment
- Espace de stockage dimensionné au maximum
- Calcul sur des sous-matrices
- ...

La fonction `cblas_dgemv` doit connaître :

- le nombre de colonnes de la matrice
- le nombre de colonnes de l'espace de stockage

→ Pour savoir combien d'espace sauter entre deux lignes

Ex : en *row major* sans transposer A et B :  
il faut sauter  $lda - k$ , avec :  $lda \geq k, ldb \geq n, ldc \geq n$

28

---

---

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

## 5 – Espace de stockage pour *cublasDgemv*

Les espaces de stockage peuvent être plus grand que les matrices :

- Réutilisation d'un espace alloué précédemment
- Espace de stockage dimensionné au maximum
- Calcul sur des sous-matrices
- ...

La fonction `cublasDgemv` impose un stockage en *column major* :

- Sans transposer A et B il faut :  $lda \geq m, ldb \geq k, ldc \geq m$
- En transposant A il faut :  $lda \geq k$
- En transposant B il faut :  $ldb \geq n$

29

---

---

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

## 5 – Espace de stockage pour *cublasDgemv*

Spécification de `lda` et `ldb`

| Parameter | device | input | Description   |
|-----------|--------|-------|---|
| A         |        | input | <type> array of dimensions $lda \times k$ with $lda \geq \max(1, m)$ if <code>transa == CUBLAS_OP_N</code> and $lda \times m$ with $lda \geq \max(1, k)$ otherwise. |
| lda       |        | input | leading dimension of two-dimensional array used to store the matrix A.  |
| B         |        | input | <type> array of dimension $ldb \times n$ with $ldb \geq \max(1, k)$ if <code>transa == CUBLAS_OP_N</code> and $ldb \times k$ with $ldb \geq \max(1, n)$ otherwise.  |
| ldb       |        | input | leading dimension of two-dimensional array used to store matrix B.  |

- Un tableau A d'un espace total de  $m \times k$  éléments suffira
- Si on n'a pas transposé A on indiquera  $lda = m$  (ou plus), Sinon on indiquera  $lda = k$  (ou plus)
- Un tableau B d'un espace total de  $k \times n$  éléments suffira
- Si on n'a pas transposé B on indiquera  $ldb = k$  (ou plus), Sinon on indiquera  $ldb = n$  (ou plus)

30

---

---

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

## 5 – Espace de stockage pour *cublasDgemv*

Spécification de *ldc*

| C          | device | in/out | <type> array of dimensions <i>ldc</i> x <i>n</i> with <i>ldc</i> >= max(1, <i>m</i> ). |
|------------|--------|--------|--|
| <i>ldc</i> |        | input  | leading dimension of a two-dimensional array used to store the matrix <i>c</i> .       |

- Un tableau C d'un espace total de  $m \times n$  double suffira
- C sera (forcément) en *column major*
- On indiquera  $ldc = m$  (ou plus)

31

---

---

---

---

---

---

---

---

---

---

---

---

CUBLAS library

- 1 – Les bibliothèques BLAS
- 2 – CUBLAS vs BLAS
- 3 – La fonction *cublasDgemv*
- 4 – Transposition de matrice
- 5 – Espace de stockage pour *cublasDgemv*
- 6 – CUBLAS avec TensorCores**
- 7 – Adresse mémoire d'un *symbol*

32

---

---

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

## 6 – CUBLAS avec TensorCores

L'appel aux TensorCores nécessite plus d'informations

- On passe par une **nouvelle fonction** CUBLAS (*cublasGemmEx()*)
- On précise les **types** des opérandes, du résultat, et des calculs internes
- On précise le **type d'algorithme** à utiliser

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

```

cublasStatus_t cublasGemmEx(
    cublasHandle_t handle,
    cublasOperation_t transa, cublasOperation_t transb,
    int m, int n, int k,
    const void *alpha,
    const void *A, cudaDataType_t Atype, int lda,
    const void *B, cudaDataType_t Btype, int ldb,
    const void *beta,
    void *C, cudaDataType_t Ctype, int ldc,
    cudaDataType_t computeType,
    cublasGemmAlgo_t algo)

```

33

---

---

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

## 6 – CUBLAS avec TensorCores

**L'appel aux TensorCores nécessite plus d'informations**

On peut fixer la précision des calculs : 16/32/64 bits      On peut utiliser des opérandes de précisions différentes

|            | computeType | Atype/Btype | Ctype      |
|------------|-------------|-------------|------------|
| Real       | CUDA_R_16F  | CUDA_R_16F  | CUDA_R_16F |
|            | CUDA_R_32I  | CUDA_R_8I   | CUDA_R_32I |
|            | CUDA_R_32F  | CUDA_R_16F  | CUDA_R_16F |
|            |             | CUDA_R_8I   | CUDA_R_32F |
|            |             | CUDA_R_16F  | CUDA_R_32F |
|            | CUDA_R_32F  | CUDA_R_32F  | CUDA_R_32F |
| CUDA_R_64F | CUDA_R_64F  | CUDA_R_64F  |            |
| Complex    | CUDA_C_32F  | CUDA_C_8I   | CUDA_C_32F |
|            | CUDA_C_32F  | CUDA_C_32F  | CUDA_C_32F |
|            | CUDA_C_64F  | CUDA_C_64F  | CUDA_C_64F |

34

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS

## 6 – CUBLAS avec TensorCores

**L'appel aux TensorCores nécessite plus d'informations**

- On peut utiliser ou pas les TensorCores
- On peut **fixer l'algorithme** de multiplication à utiliser ou **laisser une heuristique le choisir** (fonction des données)

| CublasGemmAlgo_t  | Meaning   |
|---|---|
| CUBLAS_GEMM_DEFAULT   | Apply Heuristics to select the GEMM algorithm   |
| CUBLAS_GEMM_ALGO0 to CUBLAS_GEMM_ALGO23                     | Explicitly choose an algorithm  |
| CUBLAS_GEMM_DEFAULT_TENSOR_OP                               | Apply Heuristics to select the GEMM algorithm while allowing the use of Tensor Core operations if possible  |
| CUBLAS_GEMM_ALGO0_TENSOR_OP to CUBLAS_GEMM_ALGO15_TENSOR_OP | Explicitly choose a GEMM algorithm allowing it to use Tensor Core operations if possible, otherwise falls back to cublas<t>gemmBatched based on computeType |

35

---

---

---

---

---

---

---

---

---

---

CUBLAS library

- 1 – Les bibliothèques BLAS
- 2 – CUBLAS vs BLAS
- 3 – La fonction *cublasDgemm*
- 4 – Transposition de matrice
- 5 – Espace de stockage pour *cublasDgemm*
- 6 – CUBLAS avec TensorCores
- 7 – Adresse mémoire d'un *symbol*

36

---

---

---

---

---

---

---

---

---

---

Bibliothèques BLAS et CUBLAS



## 7 – Adresse mémoire d'un *symbol*

**Extraction de l'adresse d'un symbole**

```

__device__ float GPU_A[N][N];
float *ptGPU_A = NULL;
cudaError_t res;
res = cudaGetSymbolAddress((void **) &ptGPU_A, GPU_A);
if (res != cudaSuccess) {.....}

```

 Perception d'une table 2D  
 En C : stockage en ligne

Ensuite : le symbole peut être manipulé comme une variable GPU

```

cudaMemcpy(ptGPU_A, ptCPU_A, sizeof(float)*m*k,
           cudaMemcpyHostToDevice);

cublasSgemv(cublasHandle,
            CUBLAS_OP_T, CUBLAS_OP_T,
            m, n, k,
            adrAlpha, ptGPU_A, lda, ptGPU_B, ldb,
            adrBeta, ptGPU_C, ldc);

```

37

---

---

---

---

---

---

---

---

CUBLAS library

# FIN

38

---

---

---

---

---

---

---

---