POLYTECH PARIS-SACLAY

GP-GPU

Bibliothèque CUBLAS

Stéphane Vialle

CentraleSupélec université PARIS-SACLAY — École doctorale Sciences et technologies de l'information et de la communication (STIC) — LISN

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

1

POLYTECH PARIS-SACLAY

Bibliothèque CUBLAS

2

Les bibliothèques « BLAS »

BLAS : Basic Linear Algebra Subprograms

- Ensemble de fonctions d'algèbre linéaire
- Prototypes publiés en 1979
  https://fr.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms
- 3 niveaux de BLAS :
  - niveau 1 : opérations sur les vecteurs
  - niveau 2 : opérations de type matrice – vecteur
  - niveau 3 : opérations de type matrice – matrice
- **UNE API standardisée et DES implantations**
  - implantations open-sources
    ex : OpenBLAS, ATLAS (très efficaces)
  - implantations propriétaires ciblées sur un type de matériel
    ex : bibliothèque MKL d'Intel
  - installer en recompilant sur la machine cible

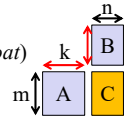
3

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*

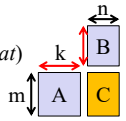
4

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

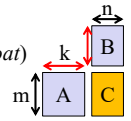
5

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

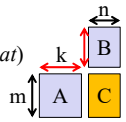
6

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)

Bibliothèque 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 !

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
```

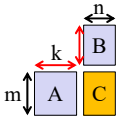
Rmq : L'initialisation (`cublasCreate`) peut prendre un peu de temps! Ne la faire qu'une seule fois au début du pgm.

Bibliothèque CUBLAS

3 – La fonction *cublasDgemm*

API :

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




```
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)
```

Comparé aux BLAS :
- On passe le « *handle* » des cublas (1er 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 :

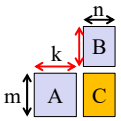
  → le stockage « *column major* » est imposé

3 – La fonction *cublasSgemm*

API :

```
cublasStatus_t cublasSgemm(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)
```

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



Comparé aux BLAS :

- On passe le « *handle* » des cublas (1er 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

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,.....)
```

A est une matrice :

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

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

→ matrice conjuguée

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

14

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,.....)
```

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 carrée (avec `transa = CUBLAS_OP_T`) on l'obtient en *column major* !

Sinon : transposition explicite à réaliser (dans une opération séparée)

15

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 (matrices carrées)

16

Bibliothèque CUBLAS

17

4 – Transposition de matrice

Sol 1 : kernel de transposition basique

```
__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];
}
```

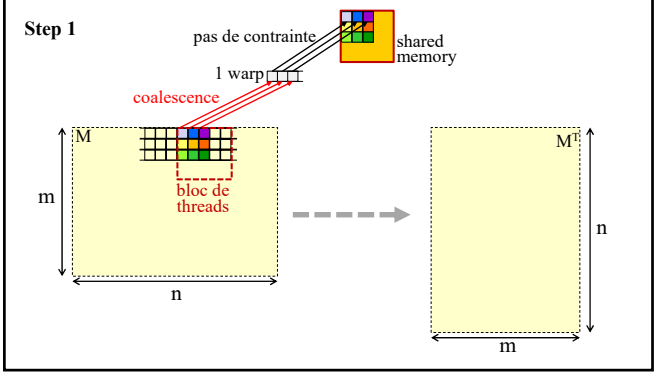
Accès NON coalescent — Accès coalescent

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

18

4 – Transposition de matrice

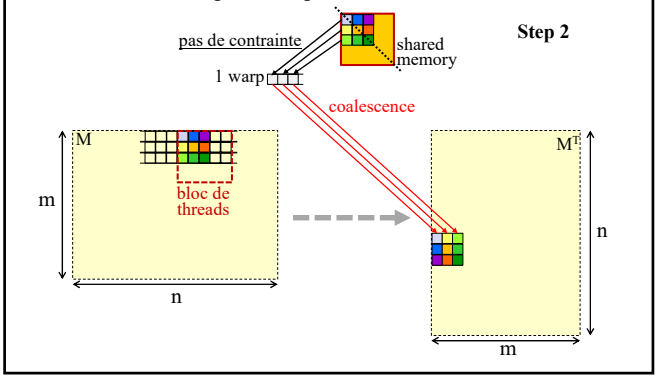
Sol 1 : kernel de transposition optimisé



19

4 – Transposition de matrice

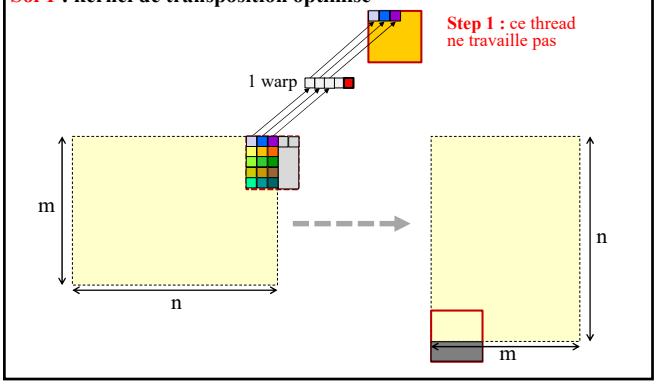
Sol 1 : kernel de transposition optimisé



20

4 – Transposition de matrice

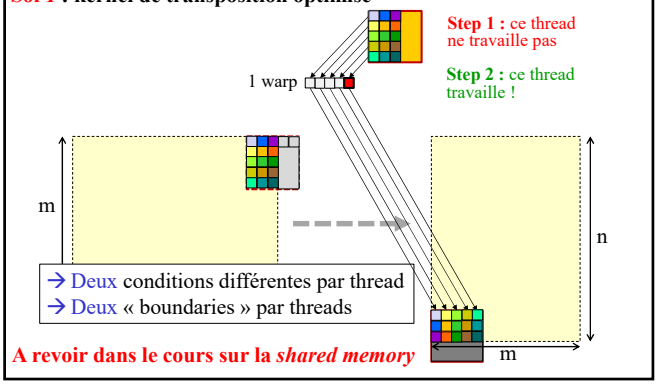
Sol 1 : kernel de transposition optimisé



21

4 – Transposition de matrice

Sol 1 : kernel de transposition optimisé



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

A revoir dans le cours sur la *shared memory*

22

4 – Transposition de matrice

Sol 1 : kernel de transposition optimisé

```
__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

4 – Transposition de matrice

Sol 2 : utilisation de `cublasDgeam` :

Réalise le calcul de : C = α.op(A) + β.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 se sert pas d'une matrice et qu'on ne fait pas d'opération dessus

➡ TP 2

24

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×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*

Bibliothèque CUBLAS

5 – Espace de stockage pour *cublasDgemm*

Autres paramètres (doc NVIDIA)

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

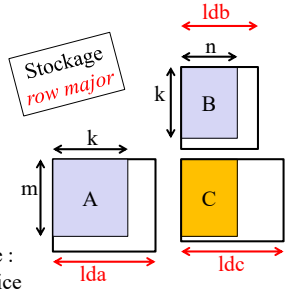
On doit préciser leur dimension principale…

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(B) 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 `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.
beta	host or device	input	<type> scalar used for multiplication. If `beta==0`, C does not have to be a valid input.
C	device	in/out	<type> array of dimensions `ldc x n` with `ldc>=max(1,m)`.
ldc		input	leading dimension of a two-dimensional array used to store the matrix C.

5 – Espace de stockage pour *cublasDgemm*

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_dgemm` doit connaitre :

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

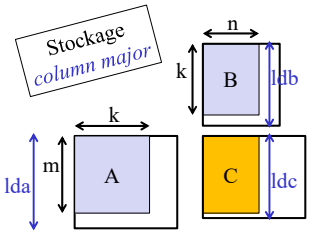
→ **Pour savoir combien d'espace sauter entre deux lignes**

Ex : en *row major* sans transposer A et B :
il faut sauter lda – k, avec : lda ≥ k, ldb ≥ n, ldc ≥ n

5 – Espace de stockage pour *cublasDgemm*

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 `cublasDgemm` impose un stockage en *column major* :

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

5 – Espace de stockage pour *cublasDgemm*

Spécification de `lda` et `ldb`

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.

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

5 – Espace de stockage pour *cublasDgemm*

Spécification de `ldc`

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

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

Bibliothèque CUBLAS

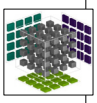
6 – CUBLAS avec TensorCores

L'appel aux TensorCores nécessite plus d'informations

- On passe par une **nouvelle fonction** CUBLAS (*cublasGemmEx()*)
- On précise les **types** des opérandes, du résultats, et des calculs internes
- On précise le **type d'algorithme** à utiliser – *deprecated on Ampere archi.*

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

```
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)
```



TP 2

6 – CUBLAS avec TensorCores

L'appel aux TensorCores nécessite plus d'info :

C = A×B :

- On peut fixer la précision des données en 16/32/64 bits (matrices A, B et C)
- On peut fixer la précision des calculs en 16/32/64 bits
- On peut travailler en Integer, Real ou Complex

Compute Type	Scale Type	Atype/Btype	Ctype
CUBLAS_COMPUTE_16F or CUBLAS_COMPUTE_16F_PEDANTIC	CUDA_R_16F	CUDA_R_16F	CUDA_R_16F
CUBLAS_COMPUTE_32I or CUBLAS_COMPUTE_32I_PEDANTIC	CUDA_R_32I	CUDA_R_8I	CUDA_R_32I
CUBLAS_COMPUTE_32F or CUBLAS_COMPUTE_32F_PEDANTIC	CUDA_R_32F	CUDA_R_16BF	CUDA_R_16BF
		CUDA_R_16F	CUDA_R_16F
		CUDA_R_8I	CUDA_R_32F
		CUDA_R_16BF	CUDA_R_32F
		CUDA_R_16F	CUDA_R_32F
		CUDA_R_32F	CUDA_R_32F
	CUDA_C_32F	CUDA_C_8I	CUDA_C_32F
		CUDA_C_32F	CUDA_C_32F
CUBLAS_COMPUTE_32F_FAST_16F or CUBLAS_COMPUTE_32F_FAST_16BF or CUBLAS_COMPUTE_32F_FAST_TF32	CUDA_R_32F	CUDA_R_32F	CUDA_R_32F
	CUDA_C_32F	CUDA_C_32F	CUDA_C_32F
CUBLAS_COMPUTE_64F or CUBLAS_COMPUTE_64F_PEDANTIC	CUDA_R_64F	CUDA_R_64F	CUDA_R_64F
	CUDA_C_64F	CUDA_C_64F	CUDA_C_64F

Débraye les tensor-cores !

6 – CUBLAS avec TensorCores

L'appel aux TensorCores nécessite plus d'info :

C = A×B :

- On peut utiliser des types standards : CUDA_R_32F, CUBLAS_COMPUTE_32F
- Ou des types faits pour des calculs rapides sur TC en simple précision mais avec une précision réduite : CUBLAS_COMPUTE_32F_FAST_TF32

Compute Type	Scale Type	Atype/Btype	Ctype
CUBLAS_COMPUTE_16F or CUBLAS_COMPUTE_16F_PEDANTIC	CUDA_R_16F	CUDA_R_16F	CUDA_R_16F
CUBLAS_COMPUTE_32I or CUBLAS_COMPUTE_32I_PEDANTIC	CUDA_R_32I	CUDA_R_8I	CUDA_R_32I
CUBLAS_COMPUTE_32F or CUBLAS_COMPUTE_32F_PEDANTIC	CUDA_R_32F	CUDA_R_16BF	CUDA_R_16BF
		CUDA_R_16F	CUDA_R_16F
		CUDA_R_8I	CUDA_R_32F
		CUDA_R_16BF	CUDA_R_32F
		CUDA_R_16F	CUDA_R_32F
		CUDA_R_32F	CUDA_R_32F
	CUDA_C_32F	CUDA_C_8I	CUDA_C_32F
		CUDA_C_32F	CUDA_C_32F
CUBLAS_COMPUTE_32F_FAST_16F or CUBLAS_COMPUTE_32F_FAST_16BF or CUBLAS_COMPUTE_32F_FAST_TF32	CUDA_R_32F	CUDA_R_32F	CUDA_R_32F
	CUDA_C_32F	CUDA_C_32F	CUDA_C_32F
CUBLAS_COMPUTE_64F or CUBLAS_COMPUTE_64F_PEDANTIC	CUDA_R_64F	CUDA_R_64F	CUDA_R_64F
	CUDA_C_64F	CUDA_C_64F	CUDA_C_64F

Débraye les tensor-cores !

6 – CUBLAS avec TensorCores

L'appel aux TensorCores nécessite plus d'info :

C = A×B :

- Ou des types faits pour des calculs en demi-précision avec risque de débordement : CUBLAS_COMPUTE_32F_FAST_16F
- Ou des types faits pour des calculs sur 16 bits avec l'amplitude de la simple précision mais avec une précision réduite : CUBLAS_COMPUTE_32F_FAST_16BF

Compute Type	Scale Type	Atype/Btype	Ctype
CUBLAS_COMPUTE_16F or CUBLAS_COMPUTE_16F_PEDANTIC	CUDA_R_16F	CUDA_R_16F	CUDA_R_16F
CUBLAS_COMPUTE_32I or CUBLAS_COMPUTE_32I_PEDANTIC	CUDA_R_32I	CUDA_R_8I	CUDA_R_32I
CUBLAS_COMPUTE_32F or CUBLAS_COMPUTE_32F_PEDANTIC	CUDA_R_32F	CUDA_R_16BF	CUDA_R_16BF
		CUDA_R_16F	CUDA_R_16F
		CUDA_R_8I	CUDA_R_32F
		CUDA_R_16BF	CUDA_R_32F
		CUDA_R_16F	CUDA_R_32F
		CUDA_R_32F	CUDA_R_32F
	CUDA_C_32F	CUDA_C_8I	CUDA_C_32F
		CUDA_C_32F	CUDA_C_32F
CUBLAS_COMPUTE_32F_FAST_16F or CUBLAS_COMPUTE_32F_FAST_16BF or CUBLAS_COMPUTE_32F_FAST_TF32	CUDA_R_32F	CUDA_R_32F	CUDA_R_32F
	CUDA_C_32F	CUDA_C_32F	CUDA_C_32F
CUBLAS_COMPUTE_64F or CUBLAS_COMPUTE_64F_PEDANTIC	CUDA_R_64F	CUDA_R_64F	CUDA_R_64F
	CUDA_C_64F	CUDA_C_64F	CUDA_C_64F

Débraye les tensor-cores !

6 – CUBLAS avec TensorCores

L'appel aux TensorCores nécessite plus d'informations

- On peut fixer l'algorithme de multiplication à utiliser ou laisser une heuristique le choisir (fonction des données)
- Inutile à partir de l'architecture AMPERE (RTX 3090) - DEPRECATED

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

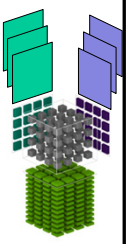
37

6 – CUBLAS avec TensorCores

Multiplication de tableaux de matrices

$$C_i = \alpha.op(A_i) \times op(B_i) + \beta.C_i$$

```
cublasGemmBatchedEx(cublasHandle_t handle,
      cublasOperation_t transa,
      cublasOperation_t transb,
      int m, int n, int k,
      const void    *alpha,
      const void    *Aarray[],
      cudaDataType_t Atype,
      int lda,
      const void    *Barray[],
      cudaDataType_t Btype,
      int ldb,
      const void    *beta,
      void          *Carray[],
      cudaDataType_t Ctype,
      int ldc,
      int batchCount,
      cublasComputeType_t computeType,
      cublasGemmAlgo_t algo)
```



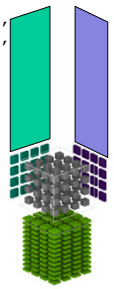
38

6 – CUBLAS avec TensorCores

Multiplication de longues suites de matrices

$$C_i = \alpha.op(A + strideA * i) \times op(B + strideB * i) + \beta.(C + strideC * i)$$

```
cublasGemmStridedBatchedEx(
      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,
      long long int strideA,
      const void    *B,
      cudaDataType_t Btype,
      int ldb,
      long long int strideB,
      const void    *beta,
      void          *C,
      cudaDataType_t Ctype,
      int ldc,
      long long int strideC,
      int batchCount,
      cublasComputeType_t computeType,
      cublasGemmAlgo_t algo)
```



39

Bibliothèque CUBLAS

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*

40

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);

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

41

Bibliothèque CUBLAS

FIN

42