ENSEA
Beyond Engineering

TP - Implémentation de CNN LeNet-5 sur GPU avec CUDA

Etudiant : Mathys BARRIE – Ousseynou NDOUR
Professeur : Nicolas LARUE

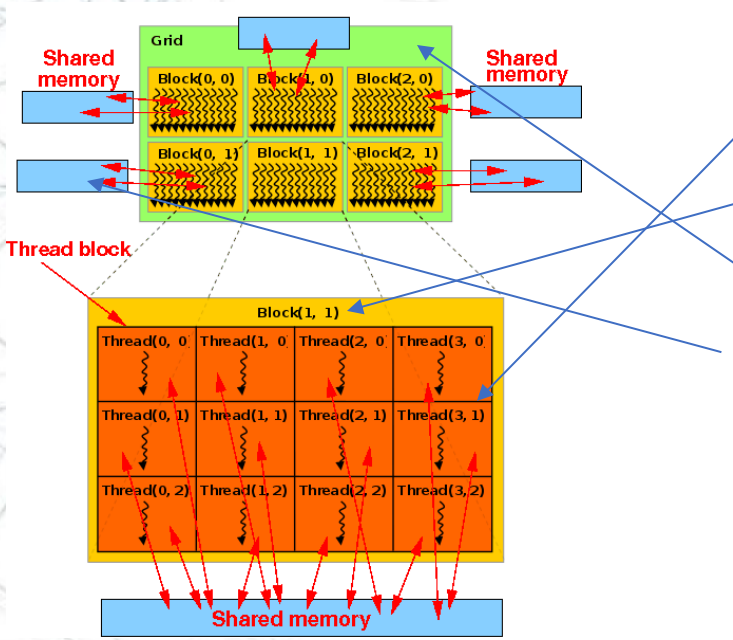
Partie 1 : Prise en main de CUDA

Objectifs principaux :

- Apprendre à utiliser CUDA pour l'accélération sur GPU.
- Comparer la complexité et les performances GPU vs CPU.
- Implémenter **l'inférence** du CNN LeNet-5.

Tâches clés :

- Prise en main des matrices : addition et multiplication.
- Analyse des performances : **temps de calcul** et **accélération**.
- Développement via **CUDA** et suivi avec **Git**.



- **Threads** : unités les plus petites exécutant les calculs.
- **Blocks** : regroupent plusieurs threads et utilisent une mémoire partagée commune.
- **Grid** : organise les blocks pour paralléliser les calculs sur tout le GPU.
- **Mémoire partagée** : essentielle pour optimiser les performances CUDA.

Ce modèle hiérarchique permet d'exploiter le **parallélisme massif** offert par les GPU pour des calculs comme la **multiplication de matrices** ou l'inférence de CNN

ENSEA
Beyond Engineering

Partie 1 : Addition et Multiplication - Temps d'exécution CPU vs GPU

```
(base) PS C:\Users\User\Desktop\Bureau\_3A ENSEA SIA\COURS\HSP\TPs\CU
Matrice 1 (partielle) :
-0.59 -0.33 0.34 0.00
0.69 0.24 -0.22 0.58
0.62 -0.36 0.05 0.45
-0.19 -0.73 0.61 -0.09
Matrice 2 (partielle) :
0.33 -0.13 -0.82 0.44
0.49 -0.20 -0.10 0.86
-0.46 0.16 -0.07 -0.55
0.25 -0.45 0.57 0.67
Temps d'ex-®cution de l'addition sur CPU : 0.001000 secondes
Temps d'ex-®cution de la multiplication sur CPU : 0.291000 secondes
Temps d'ex-®cution de l'addition sur GPU : 11.590048 ms
Temps d'ex-®cution de la multiplication sur GPU : 0.420864 ms
```

Fonctions développées :

- Addition sur CPU et GPU : *`MatrixAdd`*, *`cudaMatrixAdd`*.
- Multiplication sur CPU et GPU : *`MatrixMult`*, *`cudaMatrixMult`*.

Complexité théorique :

- Addition : O(n×p)
- Multiplication : O(n^3)

Addition :

- **GPU** (**11.59 ms**) est **plus lent** que le CPU (**1 ms**), probablement à cause de l'**overhead de transfert mémoire** entre CPU et GPU.

Multiplication :

- **GPU** (**0.42 ms**) est **beaucoup plus rapide** que le CPU (**291 ms**).

La multiplication est fortement accélérée grâce au **parallélisme** du GPU.

ENSEA
Beyond Engineering

Partie 1 : Addition et Multiplication - Temps d'exécution CPU vs GPU

Nous avons mesuré et comparé les temps d'exécution pour l'addition et la multiplication de matrices sur CPU et GPU en faisant varier la taille n x n des matrices.
Les résultats obtenus sont présentés dans le tableau suivant :

Taille n de la matrice	10	1000	10000
Temps d'exécution Addition CPU	0ms	2ms	216ms
Temps d'exécution Addition GPU	0,330ms	0,296ms	7,445ms
Temps d'exécution Multiplication CPU	2ms	2380ms	Pas de valeur **(n trop grand)**
Temps d'exécution Multiplication GPU	0.029ms	2,802ms	2789ms

Addition CPU vs GPU :

- Pour des matrices de petite taille (n=10), le GPU est légèrement plus lent que le CPU, probablement à cause des surcoûts liés au transfert des données entre le CPU et le GPU.
- À mesure que la taille de la matrice augmente, le GPU devient beaucoup plus rapide que le CPU, avec une différence marquée pour n=1000 et n=10000.

Multiplication CPU vs GPU :

- La multiplication de matrices est une opération bien parallélisable. Dès n=10, le GPU est beaucoup plus rapide que le CPU.
- Pour n=10000, le CPU ne peut pas calculer la multiplication dans un temps raisonnable, alors que le GPU gère l'opération en moins de 3 secondes.

ENSEA
Beyond Engineering

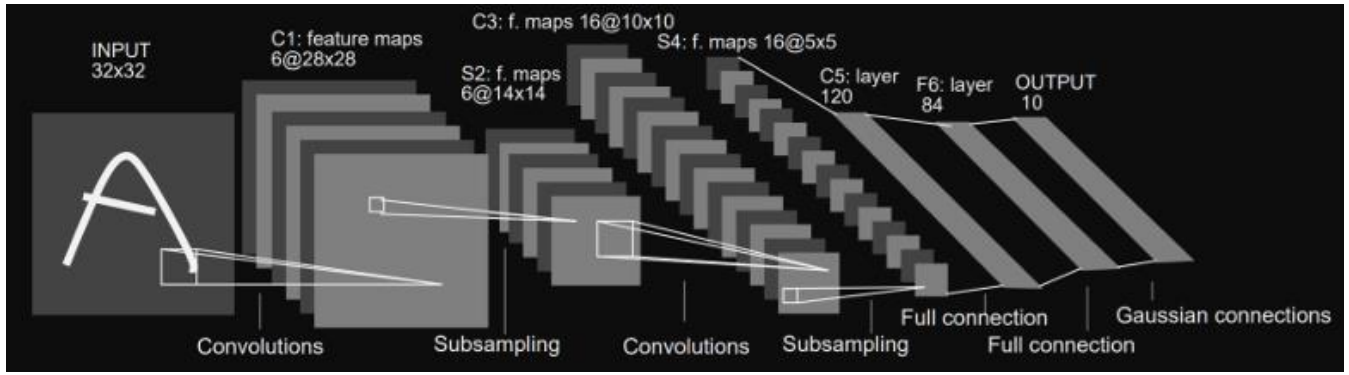
Conclusion sur la 1ère partie

- Le GPU offre une accélération significative pour les opérations lourdes sur des matrices de grande taille, comme la multiplication.
- Les surcoûts initiaux (transfert CPU → GPU et configuration des blocs) sont négligeables pour des matrices de grande taille.
- Ces résultats confirment l'intérêt de CUDA pour l'accélération des calculs intensifs, en particulier dans les tâches d'apprentissage automatique ou de traitement d'image où de grandes matrices sont courantes.

ENSEA
Beyond Engineering

Implémentation du LeNet-5 sur GPU

Le pipeline d'implémentation du LeNet-5 sur GPU est organisé pour exploiter la parallélisation des calculs dans les convolutions, les sous-échantillonnages, et les couches fully connected, permettant une accélération significative par rapport à une implémentation CPU.



1. Génération des données d'entrée

Initialisation des matrices nécessaires :

- **`raw_data`** (taille : 32×32) : Données d'entrée normalisées entre 0 et 1.
- **`C1_data`** (taille : 6×28×28) : Sortie de la première convolution.
- **`S1_data`** (taille : 6×14×14) : Sortie du sous-échantillonnage.
- **`C1_kernel`** (taille : 6×5×5) : Noyaux de convolution.

2. Convolution 2D

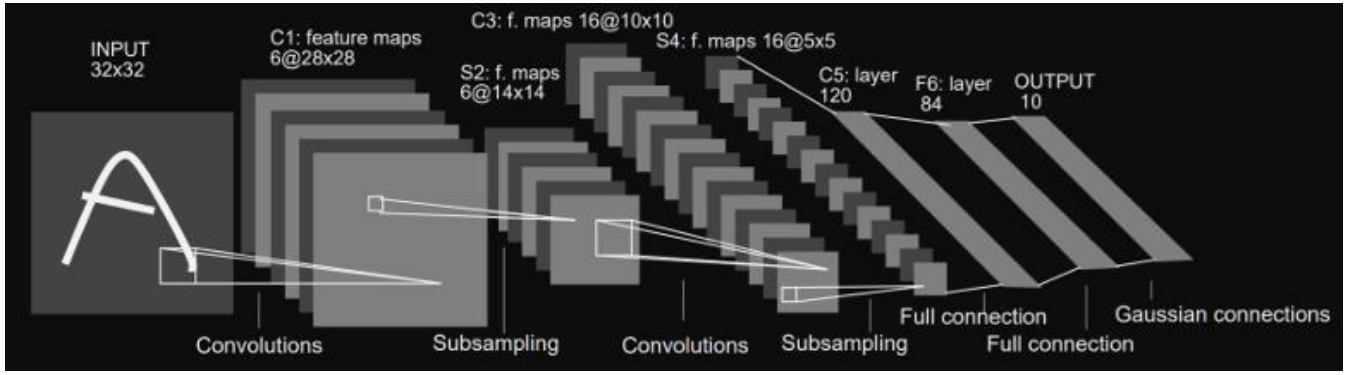
Fonction principale : **`cudaConvolution2D`**.

Étapes :

- Parcours de chaque noyau (canal par canal).
- Calcul de la somme pondérée des pixels sur la région locale.
- Application de l'activation **`tanh`** pour introduire la non-linéarité.
- Comparaison des sorties :
- Résultats conformes aux attentes après normalisation.

ENSEA
Beyond Engineering

Implémentation du LeNet-5 sur GPU



3. Sous-échantillonnage

- Méthode : Moyennage des pixels dans une région 2×2.
- Fonction utilisée : **`subsampling2D`**.
- Objectif : Réduction des dimensions pour passer de 6×28×28 à 6×14×14.

4. Couches Fully Connected

Fonctions :

- **`fullyConnected120`** : C5 (120 neurones).
- **`fullyConnected84`** : F6 (84 neurones).

Étapes :

- Calcul du produit scalaire entre l'entrée et les poids.
- Ajout des biais.
- Application de l'activation **`tanh`** pour chaque neurone.

5. Softmax

Objectif : Transformer les scores en probabilités.

Implémentation CUDA :

- Utilisation de **`exp`** pour normaliser les valeurs.
- Division par la somme totale pour obtenir des probabilités.
- Sortie : Classe prédite avec la probabilité maximale.

ENSEA
Beyond Engineering

1 - Exportation des Poids et Biais depuis Python

Objectif : Préparer les poids et biais pour leur utilisation dans l'implémentation CUDA.

Étapes :

- Exportation des poids et biais depuis le modèle Keras après entraînement.
- Stockage des données dans des fichiers binaires pour un chargement rapide.

Couches concernées : 0, 2, 5, 6, 7.

- Ces couches (Conv2D, Dense) nécessitent des **poids** (matrices de calcul) et des **biais**.
- Les autres couches (AveragePooling, Flatten) n'en ont pas besoin car elles réalisent des opérations sans paramètres appris.

2. Importation des Poids et Biais dans CUDA

Objectif : Charger les fichiers binaires dans la mémoire GPU.

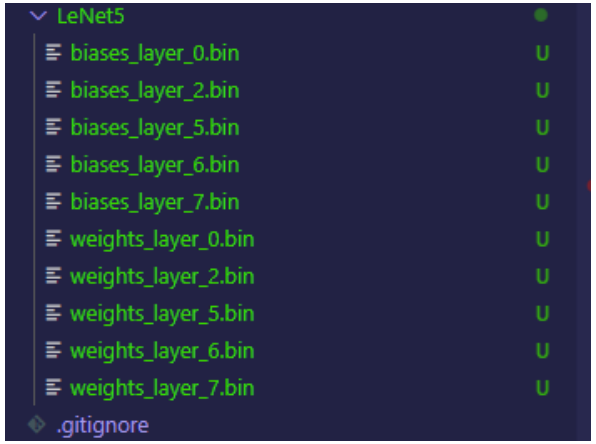
Étapes :

Lecture depuis les fichiers binaires :

- Utilisation de la fonction `loadWeights` pour charger les fichiers dans la mémoire hôte.

Transfert vers la mémoire GPU :

- Les poids et biais chargés sont transférés du CPU au GPU.



La gestion des poids et biais est essentielle pour reproduire les résultats du modèle entraîné avec précision dans CUDA.

ENSEA
Beyond Engineering

Importation et Affichage des Données MNIST

Ces 2 étapes permettent de valider la gestion correcte des données brutes, de vérifier leur intégrité visuelle, et d'assurer leur compatibilité avec CUDA, garantissant ainsi une base fiable pour le traitement dans le réseau de neurones.
Voici un exemple d'image MNIST affichée en console :

ENSEA
Beyond Engineering

Tests et Résultats

Lors de nos tests, nous avons constaté que le réseau renvoie systématiquement la classe **0**, quelle que soit l'image MNIST en entrée. Plusieurs raisons peuvent expliquer ce comportement :

Poids/biais non correspondants

- Nous utilisons peut-être des poids ou biais qui ne correspondent pas à ceux réellement entraînés (erreur d'export, inversion entre couches, dimensions incompatibles, etc.).

Problème de prétraitement

- La normalisation de nos images (division par 255, inversion des intensités, etc.) peut différer de celle utilisée lors de l'entraînement dans le notebook Python.
- Nous appliquons éventuellement un *padding* 32×32 alors que le modèle entraîné s'attendait à des images 28×28, ou inversement.

Incohérence dans l'ordre des canaux ou le Flatten

- Si, dans notre code, nous considérons l'ordre `[canaux, hauteur, largeur]`, alors que le modèle Keras attend `[hauteur, largeur, canaux]`, les multiplications pour la couche fully connected ne correspondent plus aux poids attendus.

Mauvais chargement des poids

- En cas de chargement incorrect (fichier introuvable, mauvaise taille, etc.), nos poids/biais risquent de se retrouver à zéro, ce qui biaise totalement la prédiction.

Différences d'architecture

- Le modèle original LeNet-5 utilisait un *padding* pour passer en 32×32. Si notre notebook n'a pas reproduit ce même *padding*, nos convolutions ne donnent pas les mêmes résultats que lors de l'entraînement.

ENSEA
Beyond Engineering

CONCLUSION

Dans ce TP, nous avons exploré et confirmé l'efficacité des calculs parallélisés sur GPU à travers deux parties principales :

Première Partie : Calculs sur Matrices

- Le GPU offre une accélération significative pour les opérations lourdes, comme la multiplication de grandes matrices.
- Les surcoûts initiaux (transferts CPU → GPU et configuration des blocs) deviennent négligeables pour des matrices de grande taille.
- Ces résultats valident l'intérêt de CUDA pour les calculs intensifs, notamment dans des domaines comme l'apprentissage automatique ou le traitement d'image.

Deuxième Partie : Implémentation du LeNet-5

- Nous avons implémenté les étapes clés d'un réseau convolutif en C/CUDA, depuis la lecture des données MNIST jusqu'aux prédictions.
- Les calculs convolutionnels et fully connected ont démontré le potentiel du GPU pour gérer des charges computationnelles complexes.
- Malgré des défis liés à la correspondance des poids, biais, et configurations entre Python et CUDA, nous avons identifié des axes d'amélioration pour aligner les résultats.

Ce projet nous a permis de constater la puissance du GPU pour les calculs parallèles et d'acquérir une compréhension approfondie des réseaux de neurones convolutifs. Ces compétences sont essentielles pour aborder des projets plus ambitieux en apprentissage profond ou en optimisation GPU.

ENSEA
Beyond Engineering