Etat de l'art : Étude des calculs GP-GPU sur cartes graphiques

Quentin Maurois, Julien Soto

I : Historique et architectures des GPU

Pour mieux comprendre l'avancée des cartes graphiques, nous nous sommes intéressés à l'histoire de NVIDIA, un acteur majeur du domaine, depuis sa création jusqu'à sa position dominante actuelle sur le marché des GPU.

1. Débuts de NVIDIA

NVIDIA Corporation a été fondée en 1993 par Jensen Huang, Chris Malachowsky et Curtis Priem. Son objectif initial était de révolutionner le traitement graphique en concevant des cartes graphiques destinées aux applications visuelles, notamment pour les rendus 3D et l'amélioration des performances des jeux vidéo [1].

La première carte graphique conçue par NVIDIA était la NV1, également connue sous le nom de STG2000. Bien qu'elle ait suscité un certain intérêt pour ses caractéristiques innovantes, elle n'a pas rencontré le succès commercial escompté. Cependant, cette expérience a posé les bases pour l'innovation et le développement de nouvelles technologies graphiques.

En 1999, c'est NVIDIA qui introduit le concept de GPU, Graphics Processing Unit, avec le lancement de la GeForce 256. Cette carte a été la première à intégrer des fonctionnalités de traitement graphique avancées, notamment la transformation et l'éclairage matériels, Transform and Lighting, définissant ainsi une nouvelle norme pour l'industrie. Le GPU est rapidement devenu un élément clé non seulement pour les jeux vidéo, mais également pour d'autres domaines comme la modélisation 3D, la visualisation scientifique et, plus tard, l'intelligence artificielle [8].

2. Architectures GPU

Depuis sa création, NVIDIA a conçu plusieurs architectures GPU, chacune marquée par des avancées technologiques significatives. Ces évolutions ont contribué à des améliorations en matière de performance, d'efficacité énergétique et de fonctionnalités [2] [3].

- Fahrenheit (1995) : Utilisée dans les premiers GPU, elle a permis d'introduire une meilleure gestion des rendus graphiques 3D.
- Celsius (2000) : Une architecture qui a apporté des innovations dans le traitement de la géométrie et des textures.
- Kelvin (2001) : Conçue pour offrir de meilleures performances en calculs vectoriels.
- Rankine et Curie (2002-2004) : Introduisent les shaders programmables.
- Tesla (2006) : Une architecture qui a permis de poser les bases pour les calculs parallèles massifs, introduisant CUDA pour les applications scientifiques et industrielles [10].

- Fermi (2010) : L'architecture Fermi, avec la série GTX 400, a apporté des améliorations significatives en matière de parallélisme et de précision.
- Kepler (2012) : L'architecture Kepler a été conçue pour une meilleure efficacité énergétique et a été introduite avec les GTX 600 et 700 [11].
- Maxwell (2014) : Avec les cartes GTX 900, Maxwell a optimisé l'utilisation de l'énergie et a introduit des fonctions avancées de rendu.
- Pascal (2016) : Les cartes GTX 10xx ont été les premières à exploiter la gravure en 16 nm et à proposer des performances accrues pour la réalité virtuelle (VR) [7].
- Volta (2017) : Conçue principalement pour les calculs IA, elle introduit les Tensor Cores pour accélérer les opérations de deep learning [7].
- Turing (2018) : Avec les cartes RTX 20xx, elle a révolutionné le rendu graphique avec le ray tracing en temps réel et le DLSS (Deep Learning Super Sampling) [7].
- Ampere (2020) : L'architecture Ampere, présente dans les RTX 30xx, a doublé les performances graphiques et amélioré les calculs IA [7].
- Ada Lovelace (2022) : Lancée en 2022 avec les RTX 40xx, cette architecture introduit les cœurs Tensor de 4e génération et des moteurs de ray tracing améliorés, idéale pour le calcul intensif [12].

L'évolution des GPU de NVIDIA repose sur des avancées matérielles majeures. L'augmentation constante du nombre de transistors, rendue possible par des processus de gravure de plus en plus fins (de 130 nm à 5 nm), a permis de décupler les performances et d'augmenter la densité des circuits. En parallèle, les innovations dans la mémoire vidéo ont significativement accrû la bande passante et amélioré la gestion des données graphiques. Par ailleurs, l'augmentation du nombre de cœurs CUDA a transformé les GPU en puissants outils de calcul parallèle, idéaux pour les jeux, les applications graphiques et l'intelligence artificielle.

3. Différences entre architectures GPU et CPU

Les unités centrales de traitement (CPU) et les unités de traitement graphique (GPU) diffèrent fondamentalement dans leur architecture et leur optimisation pour divers types de calculs.

Les unités centrales de traitement sont conçues pour le traitement séquentiel. Une CPU comporte généralement un nombre restreint de cœurs puissants capables de gérer une variété de tâches complexes. Cette architecture est idéale pour des opérations nécessitant des calculs séquentiels rapides et une gestion efficace des tâches multiples.

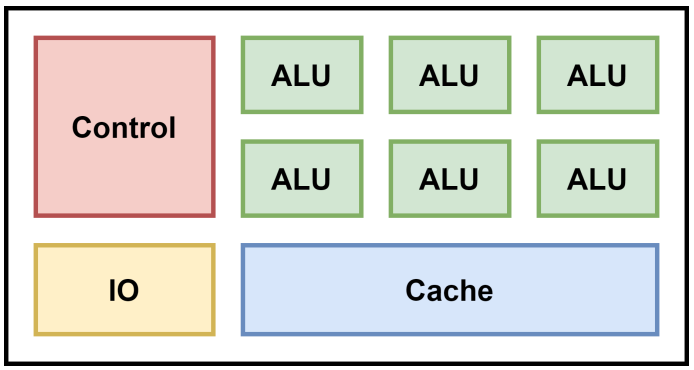

Figure 1: CPU Architecture [4]

Les unités de traitement graphique sont optimisées pour le parallélisme massif. Une GPU possède des milliers de cœurs plus simples qui exécutent simultanément la même opération sur de multiples données. Cette conception est particulièrement efficace pour des tâches hautement parallélisables, comme le rendu graphique et les calculs scientifiques intensifs.

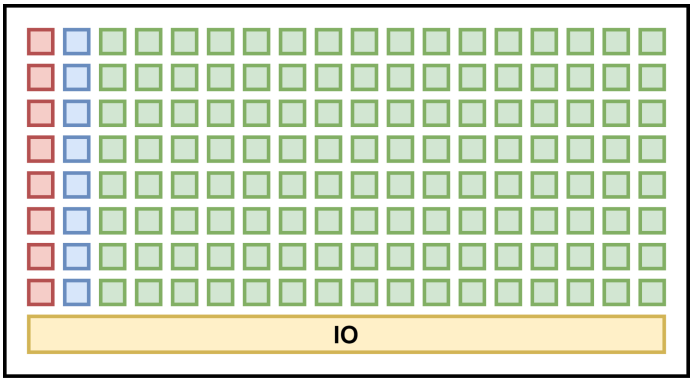

Figure 2 : GPU Architecture [4]

Les GPU excellent dans les opérations matricielles, un élément central de l'apprentissage profond et du calcul scientifique, grâce à leur architecture optimisée pour l'exécution parallèle. En particulier, leur capacité à effectuer des multiplications matricielles en parallèle leur permet souvent de surpasser les CPU en termes de performances pour ces tâches spécifiques [9].

Dans le domaine de l'apprentissage profond, les GPU jouent un rôle crucial tant pour l'entraînement que pour l'inférence des modèles. Elles sont idéales pour traiter simultanément de vastes ensembles de données, ce qui réduit considérablement les temps de calcul pour des réseaux neuronaux complexes. En revanche, les CPU conservent leur importance pour des tâches impliquant une logique séquentielle ou une gestion complexe, comme le prétraitement des données et la coordination des processus.

Lorsque l'on compare les performances des GPU et des CPU dans l'entraînement des réseaux neuronaux, les GPU offrent un avantage significatif en termes de rapidité, particulièrement pour les réseaux profonds et complexes. Cependant, pour des réseaux plus simples ou des tâches moins intensives, l'écart de performances entre GPU et CPU peut être moins prononcé.

Malgré leurs avantages en matière de parallélisme, les GPU consomment généralement plus d'énergie que les CPU. Ce facteur rend essentiel l'évaluation du rapport performance/consommation énergétique lors du choix d'une architecture pour une application spécifique. Un équilibre entre efficacité énergétique et performances est déterminant pour optimiser les ressources dans des projets de calcul intensif.
[5]

II. CUDA et OpenCL - Comparaison des API de programmation GPU

1. Présentation des deux technologies

CUDA est l'API propriétaire développée par NVIDIA pour la programmation parallèle sur GPU. Elle se limite aux cartes de la marque et est conçue pour tirer parti de leur architecture au travers de différents outils et bibliothèques dédiés. Cette API est utilisée dans différents domaines tels que le calcul scientifique, le deep learning ou encore les simulations physiques [13].

Le principal inconvénient de cette API est sa dépendance à l'écosystème NVIDIA. Cette restriction limite fortement la portabilité des applications développées avec CUDA. De plus, la position de monopole de NVIDIA leur permet de maintenir des prix élevés pour leurs GPU, rendant ces technologies moins accessibles, ce qui peut représenter un obstacle significatif dans des domaines comme la recherche, souvent confrontée à des contraintes budgétaires.

D'un autre côté, CUDA présente de nombreux avantages, notamment ses hautes performances sur les GPU NVIDIA grâce à son optimisation spécifique. Son environnement de développement complet, incluant des bibliothèques optimisées (cuBLAS, cuFFT) et des outils comme Nsight, facilite le travail des développeurs et réduit le temps nécessaire à l'écriture et au débogage de code parallèle. De plus, la documentation détaillée et le large support de la communauté renforcent l'attractivité de cette API, en particulier pour les projets exigeants tels que le deep learning, les simulations physiques et les calculs scientifiques.

OpenCL, de son côté, est une API de calcul parallèle ouverte développée par le groupe Khronos. Elle est conçue pour fonctionner sur différentes plateformes, pas seulement des GPUs mais également des CPU, FPGA (Field Programmable Gate Arrays) ou d'autres accélérateurs. Cette flexibilité permet de créer des applications portables et performantes [14].

Bien que sa flexibilité soit un atout majeur, OpenCL ne bénéficie pas d'une optimisation spécifique pour chaque matériel supporté, ce qui peut entraîner des performances inférieures à celles de solutions propriétaires comme CUDA. De plus, le support de certains matériels, bien qu'annoncé comme compatible, peut s'avérer limité en termes de performances ou même incomplet, ce qui constitue un frein à son adoption dans certains contextes exigeants.

OpenCL peut être moins optimisé pour certains matériels, mais il reste performant sur des composants courants comme les GPU et CPU, avec des performances parfois comparables à CUDA sur les GPU NVIDIA. Son caractère multi-plateformes permet de l'utiliser sur divers dispositifs, ce qui le rend adapté aux environnements hétérogènes où l'interopérabilité est importante. Bien qu'il ne soit pas toujours aussi finement optimisé que CUDA pour des architectures spécifiques, il offre un bon compromis entre compatibilité et performances.

2. Différences fondamentales entre CUDA et OpenCL

En termes de performances, CUDA et OpenCL présentent des résultats distincts, chacun ayant ses avantages en fonction du contexte d'utilisation. Dans l'étude intitulée "A

Performance Comparison of CUDA and OpenCL" [6], les auteurs comparent directement les performances de CUDA et OpenCL à travers des noyaux complexes issus d'une application de Monte Carlo quantique. Bien que OpenCL soit plus flexible en permettant la programmation sur une variété de plateformes (GPU, CPU, FPGA), cette portabilité accrue peut entraîner une légère pénalité de performance par rapport à CUDA, qui est spécifiquement optimisé pour les GPU NVIDIA. L'étude montre que CUDA surpasse OpenCL dans des applications nécessitant des calculs parallèles intensifs, notamment grâce à une gestion plus fine de la mémoire et une meilleure exploitation des architectures des GPU NVIDIA. Les tests de performance mesurent les temps de transfert de données vers et depuis le GPU, ainsi que les temps d'exécution des noyaux et des applications pour les deux frameworks. Ainsi, bien que OpenCL soit plus polyvalent, CUDA reste plus performant dans des contextes spécifiques grâce à son optimisation pour les GPU NVIDIA.

La rapidité d'apprentissage et la facilité d'intégration des API CUDA et OpenCL varient considérablement en fonction des objectifs et des compétences du développeur. CUDA, étant une technologie propriétaire développée spécifiquement pour les GPU NVIDIA, bénéficie d'un écosystème de développement bien intégré, avec des outils, des bibliothèques et des documentations détaillées. Cela permet aux développeurs de se familiariser rapidement avec les concepts fondamentaux de la programmation parallèle, particulièrement dans des domaines tels que le deep learning et le calcul scientifique. De plus, CUDA offre des bibliothèques hautement optimisées (comme cuBLAS et cuFFT) qui simplifient l'implémentation de nombreuses tâches complexes. Cependant, cette spécificité d'écosystème peut aussi limiter la flexibilité pour les développeurs qui cherchent à porter leurs applications sur différentes plateformes.

En revanche, OpenCL, en tant qu'API ouverte et multi-plateformes, présente une courbe d'apprentissage généralement plus abrupte. Sa flexibilité permet de cibler une grande variété de matériels, mais cela nécessite une gestion plus complexe des ressources matérielles et des optimisations spécifiques à chaque plateforme. OpenCL est conçu pour fonctionner sur des architectures hétérogènes, ce qui implique une approche plus générique et moins optimisée que celle de CUDA. L'intégration d'OpenCL dans un projet peut donc s'avérer plus complexe, en particulier dans des environnements où des optimisations matérielles poussées sont nécessaires. Cependant, pour les projets qui requièrent une portabilité entre différentes architectures, OpenCL reste un choix pertinent, bien qu'il demande davantage de temps pour parvenir à une performance optimale.

3. Applications typiques

CUDA est privilégiée dans les situations où des performances maximales sur des GPU NVIDIA sont essentielles, en particulier pour des tâches nécessitant une optimisation fine du matériel. Cela inclut des domaines comme le deep learning, la simulation scientifique et les calculs numériques intensifs. Par exemple, les frameworks de deep learning comme TensorFlow ou PyTorch utilisent largement CUDA pour accélérer l'entraînement de réseaux neuronaux, profitant des optimisations spécifiques aux GPU NVIDIA pour réduire les temps d'exécution. De même, dans les simulations physiques complexes ou les analyses de données volumineuses, CUDA permet de tirer pleinement parti des capacités de parallélisme des GPU NVIDIA, offrant ainsi des gains de performance significatifs. Enfin, CUDA bénéficie d'un large éventail de bibliothèques et d'outils dédiés, ce qui facilite l'intégration dans des applications nécessitant des calculs intensifs.

OpenCL est utilisé dans les cas où l'environnement n'utilise pas de matériel NVIDIA.

On le retrouve fréquemment dans les systèmes embarqués, où il permet d'exploiter des ressources hétérogènes, telles que des CPU, GPU d'autres fabricants (comme AMD) ou des FPGA. Grâce à sa portabilité, OpenCL est souvent préféré dans des configurations multi-plateformes où l'objectif est d'assurer une compatibilité avec différents types de matériels. Cela permet de développer des applications qui peuvent s'exécuter sur une grande variété de dispositifs, ce qui est particulièrement utile dans des environnements à ressources limitées ou des systèmes distribués.

III. Résultats des recherches bibliographiques et analyses

1. Résultats des recherches sur l'évolution des GPU

Les recherches que nous avons effectuées en amont sur l'évolution des GPU se sont principalement appuyées sur les listes des cartes graphiques développées par NVIDIA [15]. Cette exploration a permis de mettre en lumière les différents types d'architectures ayant marqué l'histoire des GPU NVIDIA, ainsi que l'amélioration progressive des performances et des concepts technologiques associés.

Les Figures 3, 4 et 5 que nous avons réalisées à partir de l'analyse des données compilées dans ces listes, illustrent ces évolutions. Figure 3 compare les horloges mémoires et de cœurs entre différents GPU NVIDIA, tandis que Figure 4 présente la taille de mémoire moyenne et Figure 5 les performances moyennes en FP32 selon l'architecture GPU.

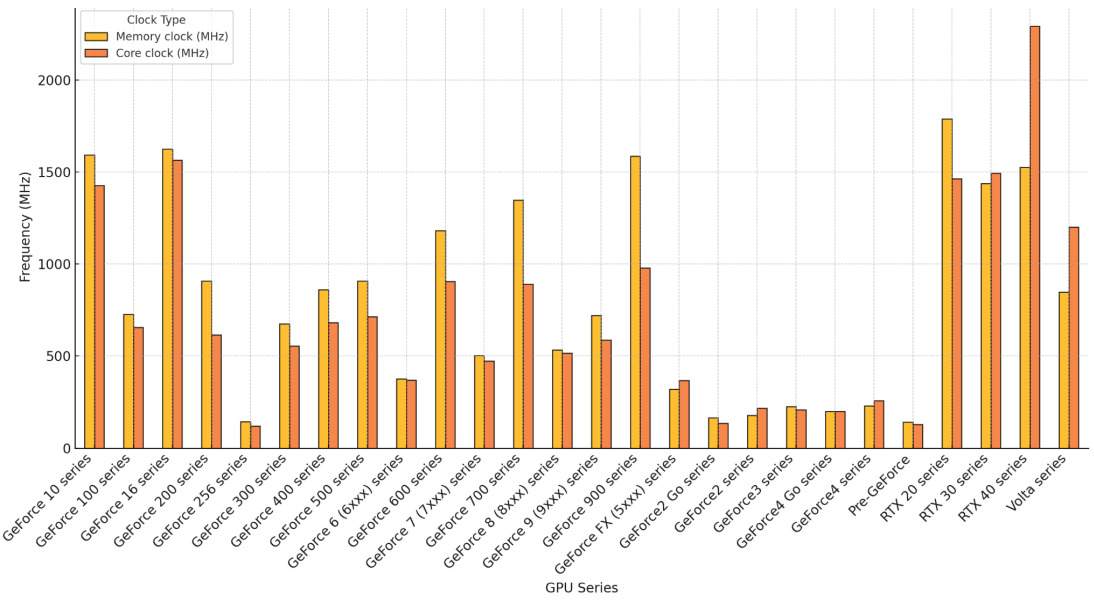

Figure 3 : Comparaison des horloges mémoires et de coeurs entre différents GPU de NVIDIA

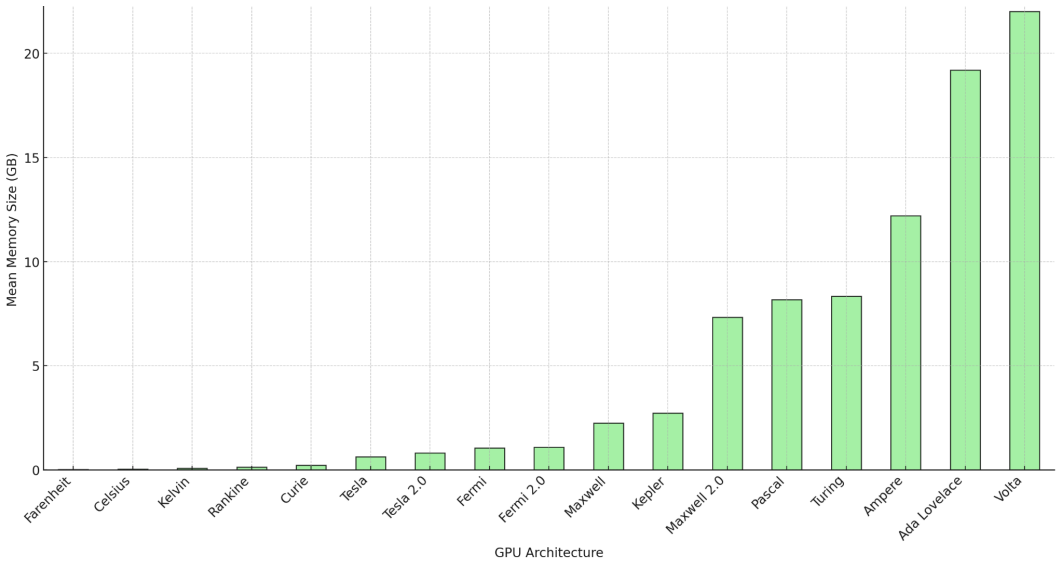

Figure 4 : Taille de mémoire moyenne selon l'architecture GPU

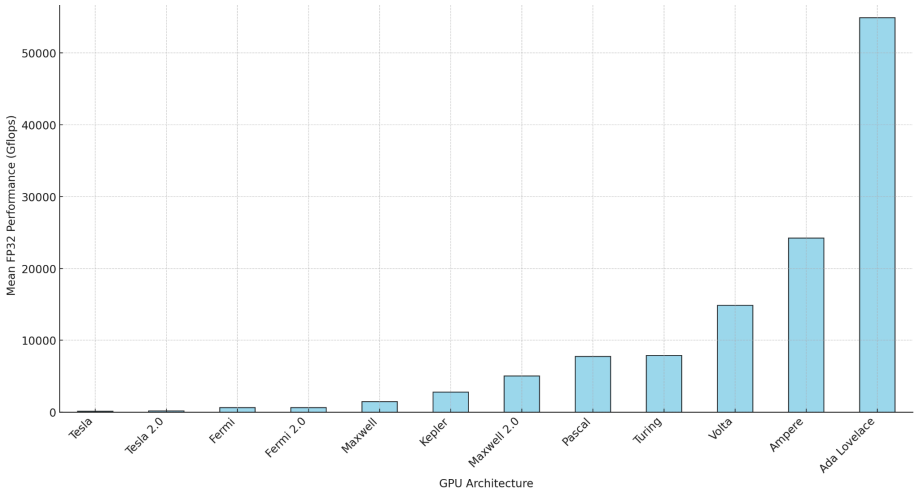

Figure 5 : Performance moyenne FP32 selon l'architecture GPU

En analysant ces listes, une logique dans la nomenclature des modèles a également pu être déterminée. Les numéros associés aux modèles servent à introduire de nouvelles technologies ou architectures, signalant une avancée significative dans la gamme. Les lettres jointes, telles que TI, RTX, GT, GTX ou TITAN, apportent des précisions sur les performances ou les applications spécifiques.

Par exemple, les modèles RTX indiquent la prise en charge du ray tracing tandis que TITAN cible les applications professionnelles nécessitant une puissance extrême. Cette structure de dénomination reflète une hiérarchie claire, facilitant la compréhension des caractéristiques et des usages pour les consommateurs et les professionnels.

2. Difficultés méthodologiques rencontrées

Les recherches ont été confrontées à plusieurs difficultés, notamment la rareté des sources fiables et accessibles concernant les premières générations d'architectures et de cartes graphiques. En effet, les documents officiels ou les archives détaillées sur les modèles initiaux sont peu nombreux, ce qui complique la reconstitution précise de leur évolution. Cette lacune a souvent conduit à une dépendance à des sources moins académiques telles que des blogs spécialisés, des forums de discussion, et des plateformes collaboratives comme Wikipédia. Bien que ces sources puissent fournir des pistes intéressantes et des détails historiques, leur vérifiabilité et leur exhaustivité restent limitées, imposant une prudence particulière dans l'interprétation des données recueillies. Cette contrainte a également nécessité un croisement minutieux des informations pour s'assurer de leur cohérence et fiabilité.

Dans l'optique de réaliser des benchmarks sur les apis CUDA et OpenCL nous voulions créer une machine virtuelle afin d'avoir un environnement de test réplicable et facile à déployer pour l'ensemble des expérimentations. Cependant, nous avons rencontré des difficultés majeures lors de la mise en place de cette VM, notamment en raison des performances limitées de l'infrastructure disponible. Cela a entravé notre capacité à tester efficacement les environnements CUDA et OpenCL dans des conditions réalistes.

Par la suite, nous avons dû installer les environnements directement sur des machines physiques, mais nous avons alors fait face à des problèmes de compatibilité entre les versions du CUDA Toolkit et des drivers NVIDIA. En effet, certaines versions récentes du CUDA Toolkit ne sont pas compatibles avec des versions plus anciennes des drivers NVIDIA, ce qui a généré des erreurs et des instabilités. Cette incompatibilité a nécessité plusieurs tentatives de réinstallation et de mise à jour des drivers pour résoudre les conflits, ce qui a prolongé de manière significative le temps nécessaire à la mise en place des environnements.

Plan d'avancement pendant la période à plein temps

- Effectuer différents types de benchmarks
- Adaptation et optimisation des algorithmes pour l'exécution sur GPU avec CUDA et OpenCL.
- Exécution des benchmarks sur différents types de calculs (matriciels, prédictifs), collecte et analyse des données de performances.
- Compilation des résultats, finalisation du rapport comparatif, et préparation d'une présentation des résultats obtenus.

Plan prévisionnel du rapport

1. Introduction
   État d'avancement : à mettre au propre
2. État de l'art
   État d'avancement : à valider
3. Méthodologie
   État d'avancement :
   - Installation de l'environnement CUDA sous Windows

   Reste à faire :
   - Installation des environnements CUDA et OpenCL sous linux
   - Finalisation de la définition des benchmarks (types d'opération à réaliser, tailles des matrices)
   - Finalisation de la définition des mesures à collecter (temps d'exécution, consommation énergétique, utilisation des ressources, bande passante, …)
   - Exécution des benchmarks

4. Résultats et analyse
   État d'avancement :
   non commencé, pas encore de résultat à analyser
5. Discussion et conclusion
   État d'avancement : pas de résultats

6. Annexes
   État d'avancement :
   Feuille de calculs contenant les spécificités techniques des différentes cartes graphiques analysées

Références bibliographiques

[1] NVIDIA “About us” https://www.nvidia.com/en-us/about-nvidia

[2] NVIDIA “Graphics Cards by GeForce” https://www.nvidia.com/en-us/geforce/graphics-cards/

[3] TechPowerUp “GPU Specs Database” https://www.techpowerup.com/gpu-specs/

[4] Ming Li, Ziqian Bi, Tianyang Wang, Yizhu Wen, Qian Niu, Junyu Liu, Benji Peng, Sen Zhang, Xuanhe Pan, Jiawei Xu, Jinlang Wang, Keyu Chen, Caitlyn Heqi Yin, Pohsun Feng, Ming Liu “Deep Learning and Machine Learning with GPGPU and CUDA: Unlocking the Power of Parallel Computing” https://arxiv.org/pdf/2410.05686v2

[5] L.A. Torres, Carlos J. Barrios H., Yves Denneulin. “Evaluation of Computational and Power Performance in Matrix Multiplication Methods and Libraries on CPU and GPU using MKL, cuBLAS, and SYCL”. https://arxiv.org/pdf/2405.17322

[6] Kamran Karimi, Neil G. Dickson, Firas Hamze. A Performance Comparison of CUDA and OpenCL. https://arxiv.org/vc/arxiv/papers/1005/1005.2581v1.pdf

[7] Marek Toma Masaryk University “Evolution of Nvidia GPU from microarchitectures Pascal to Ampere” https://is.muni.cz/th/auc3k/BP.pdf

[8] Chris McClanahan Georgia Tech College of Computing ”History and Evolution of GPU Architecture” https://picture.iczhiku.com/resource/paper/WyiWoZOZoDZWgvxN.pdf

[9] Shinpei Kato, Karthik Lakshmanan, and Ragunathan (Raj) Rajkumar†, Yutaka Ishikawa Department of Electrical and Computer Engineering, Carnegie Mellon University Department of Computer Science, The University of Tokyo “TimeGraph: GPU Scheduling for Real-Time Multi-Tasking Environments” https://www.usenix.org/legacy/events/atc11/tech/final_files/Kato.pdf

[10] Erik Lindholm John Nickolls Stuart Oberman John Montrym “NVIDIA TESLA: AUNIFIED GRAPHICS AND COMPUTING ARCHITECTURE” https://www.cs.cmu.edu/afs/cs/academic/class/15869-f11/www/readings/lindholm08_tesla.pdf

[11] NVIDIA “NVIDIA’s Next Generation CUDATM Compute Architecture: Kepler GK110/210“ https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf

[12] NVIDIA “NVIDIA ADA GPU ARCHITECTURE” https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf

[13] NVIDIA “CUDA Toolkit documentation 12.6 update 3” https://docs.nvidia.com/cuda/

[14] KRHONOS “Open Standard for parallel programming of heterogeneous systems” https://www.khronos.org/opencl/

[15] Wikipedia. "List of Nvidia graphics processing units."
https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units