Vers la reconfiguration adaptative de GPU pour chaque application

Alexandre Kouyoumdjian, Caroline Collange, Erven Rohou

Compas'2019 : Parallélisme/ Architecture / Système
LIUPPA - IUT de Bayonne, France, du 24 au 28 juin 2019

Vers la reconfiguration adaptative de GPU pour chaque application

Alexandre Kouyoumdjian, Caroline Collange, Erven Rohou

Inria, Univ Rennes, CNRS, IRISA
Campus de Beaulieu, 263 Avenue Général Leclerc
35042, Rennes - France
alexandre.kouyoumdjian@inria.fr, caroline.collange@inria.fr, erven.rohou@inria.fr

---

Résumé
Nous étudions les compromis dans l'agencement interne des processeurs graphiques (GPU) utilisés pour du calcul généraliste. En particulier, nous analysons l'influence de l'agencement des processeurs de flux (*Streaming Multiprocessors*) d'une architecture parallèle inspirée de NVIDIA Fermi, en vue d'estimer le potentiel d'architectures reconfigurables, permettant de basculer entre des SM étroits et nombreux d'une part, et larges et peu nombreux de l'autre, afin de mieux s'adapter à chaque application. Si les configurations étroites sont généralement plus performantes à nombre d'unités de calcul égal, elles nécessitent plus de ressources, et en particulier d'énergie. De fait, elles ne sont pas nécessairement optimales. Notre exploration de l'espace des configurations possibles montre que la largeur optimale d'un SM varie d'une application à une autre, et qu'une capacité de reconfiguration d'une application à une autre permettrait d'avoir un profil micro-architectural mieux adapté à chaque situation, avec une consommation énergétique mieux maîtrisée.

Mots-clés : GPU, SIMT, reconfiguration

---

1. Introduction

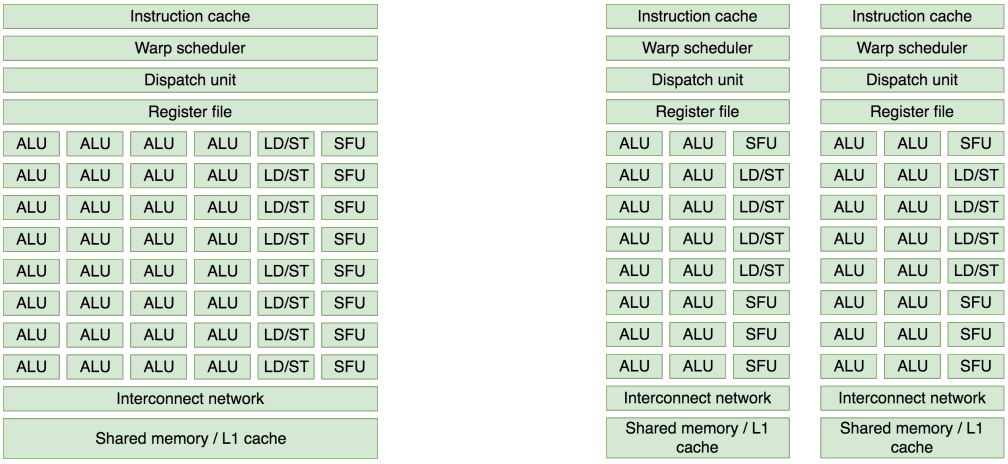
Les processeurs parallèles modernes tels que les GPU sont généralement constitués de cœurs SIMD appelés *Streaming Multiprocessors* (SM) chez NVIDIA [9], ou *Compute Units* chez AMD [12]. Ceux-ci sont constitués de logique de contrôle, de mémoire, et d'unités d'exécution regroupées au sein d'une ou plusieurs unités de type SIMD. Le passage à l'échelle d'un GPU peut donc se faire sur au moins trois dimensions : le nombre de SM, le nombre d'unités SIMD par SM, et la largeur des unités SIMD (en nombre de voies). Ces trois dimensions sont nécessaires, car le passage à l'échelle ne pourrait se faire efficacement sur une seule : élargir une unité SIMD implique notamment d'avoir un *crossbar* plus complexe entre elle et son banc de registres, et nécessite un parallélisme de données qui n'est pas toujours suffisant dans une application donnée ; augmenter le nombre d'unités SIMD par SM implique d'augmenter les ressources de contrôle par SM, et nécessite un parallélisme d'instructions qui, de même, n'est pas toujours suffisant ; enfin, augmenter le nombre de SM implique de répliquer à chaque fois toutes ses ressources de contrôle et de mémoire (ce qui a un coût élevé), et nécessite en outre un réseau d'interconnexion adapté, éventuellement très complexe et gourmand en énergie.

Pour un nombre total d'unités de calcul dans un GPU donné, différentes répartitions sur ces trois dimensions sont possibles, et ces différentes répartitions peuvent mener à différents ratios de ressources de calcul par rapport aux ressources de contrôle ou de mémoire (registres et caches). De fait, des choix différents sont faits d'une génération à l'autre [15, 6], voire au sein d'une même génération, selon la nature des applications visées par le produit final [4, 2].
Il apparaît donc intéressant, en particulier dans un contexte de calcul généraliste (n'impliquant pas le *pipeline* graphique) de pouvoir reconfigurer un même GPU d'une application à une autre pour ajuster l'agencement de ses SM afin, par exemple, d'alterner entre un mode de fonctionnement « large » avec x SM et y unités de calcul par SM, et un mode « étroit » avec $2x$ SM et $y/2$ unités de calcul par SM, comme l'illustre la figure 1. Or, dans une architecture reconfigurable, les unités nécessaires dans la configuration étroite et superflues dans la configuration large doivent néanmoins être présentes dans cette dernière. Toutefois, elles peuvent être désactivées, afin d'économiser de l'énergie.

2. Description des évaluations

2.1. Objectifs

Notre objectif est d'évaluer la sensibilité d'un GPU aux variations de paramètres de micro-architecture relatifs à la configuration des SM. Nous nous sommes penchés pour cela sur une architecture inspirée de celle de NVIDIA Fermi. Comme sur la figure 1, dans chaque SM, on trouve un cache d'instructions, un ordonnanceur, des registres, des unités de calcul, de la mémoire partagée, du cache L1, et un point d'accès au réseau d'interconnexion.



(a) Un SM large constitué de 32 ALU, 8 LD/ST, et 8 SFU.

(b) Deux SM étroits : 16 ALU, 4 LD/ST, 4 SFU chacun.

FIGURE 1 – Deux configurations possibles avec le même nombre total d'ALU, d'unités *load/store* (LD/ST), et d'unités de calcul pour les fonctions spéciales (SFU). Les ressources de contrôle doivent être répliquées. Les ressources mémoire peuvent être partitionnées, mais sont toujours répliquées dans notre étude.

2.2. Méthode

Pour évaluer l'intérêt de diverses configurations de SM, nous nous sommes appuyés sur une version du simulateur GPGPU-Sim [1] modifiée par nos soins, et une configuration dérivée d'une carte NVIDIA GeForce GTX 480. Par défaut, les 15 SM de celle-ci sont constitués dans le simulateur de deux unités SIMD-32 ; pour simplifier l'analyse des résultats, nous avons supprimé une des deux unités SIMD, et donc écarté le nombre d'unités au sein d'un SM des paramètres pouvant influer sur les résultats.
Nous avons fait varier le nombre de voies des unités SIMD, en simulant toutes les puissances de 2 comprises entre 32 et 256. Nous avons par ailleurs augmenté le nombre de SM à partir de 15, et tant que les performances augmentaient, soit jusqu'à 300 ou 420, selon les *benchmarks*. Les configurations obtenues varient donc d'un nombre total d'unités de calcul de $15 \times 32 = 480$ à $420 \times 256 = 107\,520$.

2.3. *Benchmarks* utilisés

Nous estimons ici l'intérêt potentiel d'une telle possibilité de reconfiguration. Pour ce faire, nous nous sommes intéressés à trois *benchmarks*, sélectionnés pour leurs caractéristiques très différentes, susceptibles de favoriser des configurations différentes :

- *Advanced Encryption Standard* (AES), une mise en œuvre CUDA [11] de l'algorithme de chiffrement symétrique [3] éponyme. Nous avons retenu ce *benchmark* pour sa nature très calculatoire, peu dépendante de la hiérarchie mémoire. Assez régulier, il peut en principe tirer parti d'un grand nombre de voies SIMD. Nous lui avons fait chiffrer un texte de 3 617 095 caractères, avec une clef de 256 bits.
- *Breadth-First Search* (BFS), un algorithme de parcours de graphe en largeur [14], écrit en CUDA [7]. Nous avons sélectionné ce *benchmark* pour représenter les applications très fortement dépendantes de la hiérarchie mémoire. Nous avons utilisé un graphe de 65 536 nœuds pour nos simulations.
- *Neural Network* (NN), un réseau de neurones — un type d'application de plus en plus utilisé [10]. Il fait de la reconnaissance d'images, et l'utilisateur peut spécifier le nombre d'images à traiter en parallèle. Nous avons choisi ici une valeur de 40. Nous avons retenu ce *benchmark* du fait de la popularité du *deep-learning*, qui fait l'objet d'adaptations matérielles dans les GPU récents [13] et incite au développement d'ASIC [8].

2.4. Résultats

2.4.1. *Advanced Encryption Standard*

La figure 3a présente les résultats obtenus pour AES, sous forme du nombre d'instructions par cycle (IPC). Chaque courbe représente l'IPC d'une configuration avec un nombre de voies donné par unité SIMD, en fonction du nombre total d'unités de calcul (nombre de voies $\times$ nombre de SM). Pour chaque configuration, le nombre de bancs de mémoire partagée a été porté à 256, pour éviter toute limite à ce niveau. En tout point du graphique, pour un nombre d'unités de calcul donné, les performances sont équivalentes ou meilleures avec des SM étroits, i.e. constitués d'une unité SIMD-32 ou SIMD-64. Ces deux configurations ont des performances très proches, bien supérieures à celles des configurations SIMD-128 et SIMD-256.
Cependant, une simulation menée en multipliant par 8 la fréquence du réseau d'interconnexion, du cache L2, de la mémoire, et le nombre de contrôleurs et canaux mémoire mène à des IPC de 9922, 14393, 15879 et 16674 pour les configurations SIMD-32, 64, 128 et 256, respectivement, avec 300 SM dans tous les cas (un gain d'un facteur de 10 à 17). Les configurations larges sont donc fortement limitées par la hiérarchie mémoire du GPU lorsque sa taille est très élevée. En conséquence, limitons notre interprétation des résultats de la figure 3a aux configurations

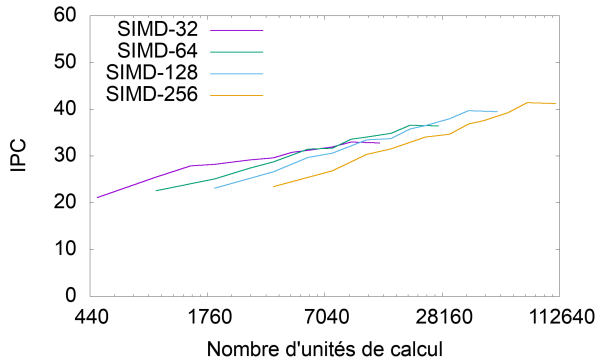
de moins de 10 000 unités de calcul : l'avantage des configurations SIMD-32 et SIMD-64 est important, légèrement à l'avantage de la première. Compte tenu de l'économie de ressources permise par une configuration SIMD-64, et de la proximité de ses performances à celles de la configuration SIMD-32, nous penchons pour la configuration SIMD-64 pour cette application, bien qu'une analyse quantitative des besoins en silicium et en énergie de chaque configuration soit désirable pour trancher de manière certaine.

2.4.2. *Breadth-First Search*

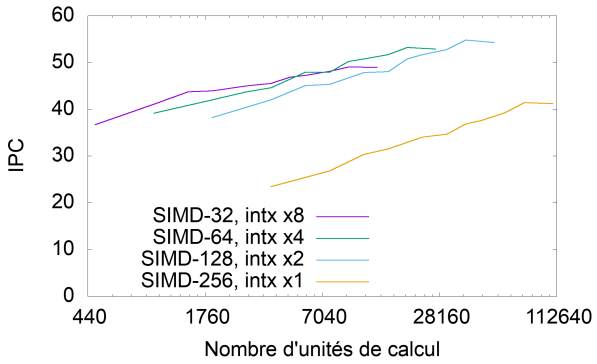
Les résultats obtenus pour BFS sont sur la figure 2a. Ici, on constate également un avantage aux configurations étroites, mais avec un écart beaucoup plus marqué entre les configurations SIMD-32 et SIMD-64, à l'avantage de la première. Ce résultat n'est pas surprenant, puisque pour un nombre d'unités de calcul égal, une configuration plus étroite et avec plus de SM a plus de cache L1 au total, et peut gérer plus d'accès mémoire simultanés. BFS étant très dépendant de la hiérarchie mémoire, ces données sont logiques. Cependant, on observe qu'à partir de 3500 unités de calcul environ, les performances des configurations les plus étroites passent moins bien à l'échelle, au point d'être dépassées par des configurations de plus en plus larges, pourtant pourvues de bien moins de SM.
Or, lorsque les SM sont très nombreux, le réseau d'interconnexion entre eux peut devenir saturé, et provoquer des ralentissements, ce qui est signalé par des *warnings* de GPGPU-Sim. C'est ce que nous avons observé ici. Pour un nombre donné d'unités de calcul, si le nombre de SM d'une configuration est doublé par rapport à une autre, il est raisonnable d'adapter les capacités du réseau d'interconnexion en conséquence. C'est pourquoi nous avons effectué les mêmes simulations, mais en multipliant par 2, 4 et 8 la fréquence d'horloge de ce réseau pour les configurations SIMD-128, 64 et 32, respectivement. Les capacités du réseau sont ainsi proportionnelles au nombre de ses nœuds, c'est-à-dire de SM. Les résultats obtenus sont sur la figure 2b.
Dès lors, la configuration SIMD-256 est largement distancée. La configuration SIMD-128, cependant, offre des résultats somme toute assez proches de ceux des configurations SIMD-32 et SIMD-64, manifestement limités par le reste du sous-système mémoire, notamment les caches, dont celui de deuxième niveau est de taille fixe, malgré l'augmentation du nombre de SM. De fait, nous ne pouvons tirer de ces résultats des conclusions trop générales, mais les configurations étroites apparaissent favorisées, pour peu que le débit de la mémoire soit adéquat. Nous tendons donc à privilégier la configuration SIMD-32 pour BFS, tout en observant que les ressources mémoire sont de toute façon primordiales.

2.4.3. *Neural Network*

Les résultats obtenus sous NN sont présentés sur la figure 3b. Ici, l'interprétation est plus simple, puisque les configurations les plus étroites sont clairement avantagées. En fait, le *benchmark* s'avère incapable de tirer parti d'une largeur d'unité SIMD supérieure à 32, de sorte que pour un nombre de SM donné, les performances sont toujours les mêmes, quelle que soit la largeur des unités SIMD. De fait, les configurations étroites sont très avantagées, car aussi performantes, mais plus économes. Bien sûr, ce comportement de NN doit être interprété comme une spécificité de ce *benchmark*, et non comme un trait commun aux réseaux de neurones artificiels, mais il a le mérite d'illustrer que des applications de ce type existent. Dans ces cas-là, la possibilité de reconfigurer un GPU pour avoir des SM plus étroits pourrait être très avantageuse, surtout s'il est impossible de modifier le code source de l'application concernée.

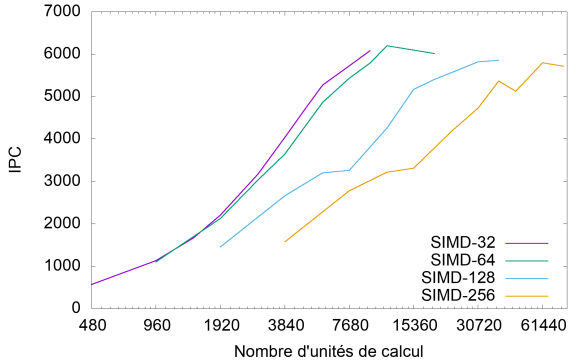


(a) Résultats sans ajustement.

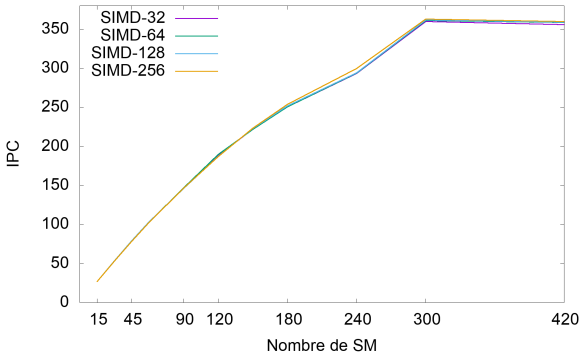


(b) Résultats avec *interconnect* ajusté

FIGURE 2 – BFS – IPC obtenus en fonction du nombre total d'unités de calcul, pour chaque largeur d'unité SIMD, sans et avec ajustement de la fréquence du réseau d'interconnexion.



(a) AES – IPC en fonction du nombre total d'ALU, pour chaque largeur d'unité SIMD.



(b) NN – IPC en fonction du nombre de SM, pour chaque largeur d'unité SIMD.

FIGURE 3

3. Conclusion

Les résultats présentés ci-dessus illustrent les différences de comportement d'une application à l'autre, en particulier concernant l'agencement optimal d'un SM. En effet, si la configuration SIMD-32 était clairement optimale pour *Neural Networks* et probablement optimale pour *Breadth-First Search*, elle ne l'est pas forcément pour *Advanced Encryption System*, plus favorable à la configuration SIMD-64. Il n'est pas exclu que d'autres applications favorisent des configurations de SM encore plus larges. Il serait donc intéressant de pouvoir reconfigurer un GPU en partitionnant les ressources d'un SM en deux, par exemple pour alterner entre une configuration SIMD-32 et SIMD-64, avant le lancement de chaque application.
En pratique, le choix de la configuration optimale pour une application donnée pourrait se faire au moins de deux façons : par une heuristique s'appuyant sur des *program features* [5], par exemple ; ou en testant les différentes configurations possibles, pour ne retenir que la plus performante.
Les avantages et inconvénients de la reconfiguration, notamment les coûts en silicium et en

énergie associés à la capacité de reconfiguration demeurent toutefois à quantifier. En effet, et comme on l'observe sur la figure 1, diviser un SM en deux nécessite non seulement de partitionner les unités de calcul, le banc de registres et les mémoires, mais aussi de répliquer l'ordonnanceur, ainsi que le point d'accès au réseau d'interconnexion. En fait, toute la logique permettant le chargement, l'ordonnancement et l'émission d'instructions doit être répliquée. Et bien qu'elle n'apparaisse pas sur la figure 1a, elle doit nécessairement y être présente, quoique désactivée. Les gains permis par la possibilité de reconfigurer la structure d'un GPU doivent donc être mis dans la balance avec ces coûts. Nous comptons donc, dans de futurs travaux, les quantifier plus précisément, afin d'évaluer l'économie d'énergie permise par les configurations les plus larges, et les éventuels gains de performance pouvant être obtenus par augmentation de la fréquence, grâce au budget énergétique libéré.

Bibliographie

1. Bakhoda (A.), Yuan (G. L.), Fung (W. W.), Wong (H.) et Aamodt (T. M.). – Analyzing cuda workloads using a detailed gpu simulator. – In *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 163–174. IEEE, 2009.
2. Choquette (J.), Giroux (O.) et Foley (D.). – Volta : Performance and programmability. *IEEE Micro*, vol. 38, n2, 2018, pp. 42–52.
3. Daemen (J.) et Rijmen (V.). – Aes proposal : Rijndael. 1999.
4. Foley (D.) et Danskin (J.). – Ultra-performance pascal gpu and nvlink interconnect. *IEEE Micro*, vol. 37, n2, 2017, pp. 7–17.
5. Fursin (G.), Kashnikov (Y.), Memon (A. W.), Chamski (Z.), Temam (O.), Namolaru (M.), Yom-Tov (E.), Mendelson (B.), Zaks (A.), Courtois (E.) et al. – Milepost gcc : Machine learning enabled self-tuning compiler. *International journal of parallel programming*, vol. 39, n3, 2011, pp. 296–327.
6. GTX (N. G.). – 980 : Featuring maxwell, the most advanced gpu ever made. *White paper, NVIDIA Corporation*, 2014.
7. Harish (P.) et Narayanan (P.). – Accelerating large graph algorithms on the gpu using cuda. – In *International conference on high-performance computing*, pp. 197–208. Springer, 2007.
8. Jouppi (N. P.), Young (C.), Patil (N.), Patterson (D.), Agrawal (G.), Bajwa (R.), Bates (S.), Bhatia (S.), Boden (N.), Borchers (A.) et al. – In-datacenter performance analysis of a tensor processing unit. – In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12. IEEE, 2017.
9. Kirk (D.) et al. – Nvidia cuda software and gpu parallel computing architecture. – In *ISMM*, pp. 103–104, 2007.
10. LeCun (Y.), Bengio (Y.) et Hinton (G.). – Deep learning. *nature*, vol. 521, n7553, 2015, p. 436.
11. Manavski (S. A.) et al. – Cuda compatible gpu as an efficient hardware accelerator for aes cryptography. *Signal Processing and Communications*, vol. 2007, 2007.
12. Mantor (M.). – Amd radeon™ hd 7970 with graphics core next (gcn) architecture. – In *Hot Chips 24 Symposium (HCS), 2012 IEEE*, pp. 1–35. IEEE, 2012.
13. Markidis (S.), Der Chien (S. W.), Laure (E.), Peng (I. B.) et Vetter (J. S.). – Nvidia tensor core programmability, performance & precision. – In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 522–531. IEEE, 2018.
14. Moore (E. F.). – The shortest path through a maze. – In *Proc. Int. Symp. Switching Theory, 1959*, pp. 285–292, 1959.
15. Wittenbrink (C. M.), Kilgariff (E.) et Prabhu (A.). – Fermi gf100 gpu architecture. *IEEE Micro*, vol. 31, n2, 2011, pp. 50–59.