Premiers résultats de comparaison des outils oneAPI et OpenCL pour la convolution 2D sur FPGA

Daouda Diakite, Nicolas Gac

Premiers résultats de comparaison des outils oneAPI et OpenCL pour la convolution 2D sur FPGA

Daouda Diakite, Nicolas Gac

Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des signaux et systèmes, 91190, Gif-sur-Yvette, France.
Daouda.Diakite@l2s.centralesupelec.fr, Nicolas.Gac@l2s.centralesupelec.fr

Résumé – Grâce aux outils de synthèse de haut niveau (HLS), les FPGA sont devenus une alternative aux GPU pour les applications à forte intensité de calcul. Ces outils ont été développés pour offrir une certaine flexibilité dans la conception des FPGA à un niveau d'abstraction plus élevé par rapport aux langages de description matérielle. Les principaux fabricants de FPGA ont proposé de nombreux outils HLS en fonction du public cible. Ce papier s'attache à comparer les outils HLS oneAPI et OpenCL en termes de performance et de productivité avec comme cas d'étude l'accélération de l'opérateur de convolution 2D. Pour des mêmes types d'optimisations (déroulage de boucle, utilisation de la mémoire locale), les performances obtenues pour les deux outils y sont présentées.

Abstract – Thanks to high-level synthesis (HLS) tools, FPGAs have become an alternative to GPUs for compute-intensive applications. These tools have been developed to provide flexibility in FPGA design at a higher abstraction level than hardware description languages. Major FPGA manufacturers have proposed many HLS tools depending on the target audience. This paper focuses on comparing the oneAPI and OpenCL HLS tools in terms of performance and productivity with the case study of 2D convolution operator acceleration. The performances obtained for both tools are presented for the same types of optimizations (loop unrolling, use of local memory).

1 Introduction

La convolution est l'un des opérateurs les plus utilisés dans les applications de traitement du signal et des images telles que la détection de contours, le floutage, la réduction de bruit [1]. C'est un opérateur très gourmand en calcul notamment utilisé dans les algorithmes itératifs de problèmes inverses comme la déconvolution [2]. Il est primordial d'avoir une puissance de calcul importante et une bande passante conséquente pour répondre à la forte demande de calcul. L'accélération de la convolution 2D a été explorée sur les différentes architectures de calcul telles que CPU, GPU ou FPGA [3, 4]. Beaucoup d'approches ont été développées pour accélérer l'opérateur de convolution telles que la réduction de la complexité arithmétique notamment par les méthodes de Winograd [5], par multiplication élément par élément dans le domaine de Fourier [6], ou par simple multiplication de matrice utilisant des tensors cores [7]. Ces travaux ont également exploré les différentes précisions de calculs (demi, fixe, simple et double) pour mieux tirer profit de l'architecture sous-jacente.

Les outils de synthèse haut niveau (HLS) ont permis la démocratisation des FPGAs à une plus large audience de programmeurs. En plus de la maturité de ces outils HLS, les FPGAs récents sont équipés de puissantes unités leurs permettant de devenir des accélérateurs de calcul pour le domaine HPC (High Performance Computing). Il n'est plus indispensable de passer par les langages de description matérielle pour utiliser les FPGAs, toutefois une connaissance approfondie de leurs architectures est nécessaire pour tirer profit de leur potentiel d'accélération. Les FPGAs ont prouvé par leur paradigme de conception leur efficacité à modéliser un pipeline sur mesure pour mieux caractériser une application et notamment les algorithmes en flot de données. La convolution est un exemple typique de traitement en flot de données à fortiori lorsqu'il s'agit d'images de flux vidéos ou les réseaux de neurone profonds. Un certain nombre de travaux dans la littérature se sont intéressés à la convolution 2D sur FPGA en utilisant les outils HLS. Un de ces travaux s'est penché sur la réduction des ressources matérielles consommées par l'implémentation de la convolution sur FPGA en utilisant le langage haut niveau CAPH [8]. Un autre exemple est l'exploration de l'espace des noyaux de convolution en appliquant plusieurs optimisations basées sur le modèle NDRange d'OpenCL sur FPGA [4].

Dans ce papier, nous comparons les outils HLS d'Intel oneAPI et OpenCL avec la convolution comme cas d'étude. OpenCL est une API de programmation parallèle open-source pour les plate-formes hétérogènes (CPU, GPU, FPGA) basée sur le standard C99. OneAPI [9] est le nouveau toolkit unifié et standardisé d'Intel qui permet de cibler plusieurs plate-formes en utilisant un seul langage de programmation C++. Intel OneAPI est basé sur le langage DPC++ qui est une extension du C++ standard incorporant le standard SYCL [10]. La terminologie utilisée pour

SYCL hérite de celle de l'outil OpenCL avec quelques ajouts spécifiques à SYCL. Par conséquent, oneAPI est à un niveau d'abstraction plus élevé à celui d'OpenCL ce qui permettrait d'améliorer la productivité en utilisant oneAPI. Par ailleurs, oneAPI étant basée sur une extension du C++, on profite aussi de la flexibilité de ce langage orienté objet. L'exploration de la convolution basée sur le parallélisme de données ayant été évaluée dans [4], ce papier s'intéressera au modèle d'exécution pipeline sur FPGA. Les optimisations classiques seront explorées telles que le pipeline et déroulage de boucle, ainsi que l'utilisation de la mémoire locale du FPGA pour augmenter le débit de calcul.

La suite de cet article est organisé comme suit : la Section 2 introduit le principe de la convolution 2D. Dans la Section 3, nous détaillons nos implémentations en présentant les différentes optimisations appliquées sur FPGA. Les résultats expérimentaux sont présentés dans la Section 4 avec une comparaison de performance entre les deux outils.

2 La convolution 2D

La convolution 2D représente une opération de multiplication accumulation entre un masque (noyau de convolution) et les pixels d'une image comme illustré sur la figure 1.

Soit f une image 2D de taille (H, W) h un masque de taille (K, K), la convolution de f par h noté F est donné par :

$$F(x,y) = \sum_{j=0}^{K-1} \sum_{i=0}^{K-1} f(x-(i-\frac{K}{2}), y-(j-\frac{K}{2}))h(i,j) \quad (1)$$

Pour gérer le problème sur les pixels de bordures, nous utilisons la méthode de bourrage de zéros (zero-padding) ce qui nous permet de conserver la taille de l'image d'entrée. La technique de bourrage de pixels est souvent la solution acceptable pour gérer les bordures d'image, et le bourrage de zéros est la plus simple à mettre en œuvre parmi les différents types de bourrage utilisés [11].

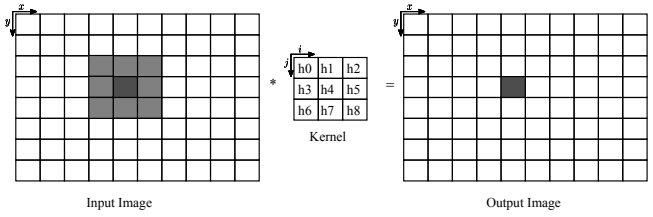

FIGURE 1 – Convolution 2D avec un masque 3×3

3 Convolution sur FPGA

Nous avons décrit l'équation 1 en OpenCL et oneAPI pour une implémentation sur FPGA en effectuant les mêmes optimisations pour évaluer l'efficacité des deux outils. Il faudra noter qu'il existe très peu de différences entre les kernels (noyau de calcul) OpenCL et oneAPI en ce qui concerne le paradigme de programmation. Toutefois, nous avons besoin, dans les deux cas, d'un code pour l'hôte qui est exécuté par le CPU et d'un code kernel qui est accéléré par le coprocesseur. L'avantage de oneAPI est qu'il est moins verbeux par rapport à OpenCL en ce qui concerne le code de l'hôte. De plus, pour oneAPI le code du kernel et de la plate-forme hôte sont dans un seul et même fichier alors qu'ils sont séparés en OpenCL. Nous allons comparer l'efficacité de nos outils sur plusieurs versions de l'opérateur de convolution.

3.1 Naïve

Cette version représente l'implémentation classique de l'opérateur (Équation 1) de manière séquentielle avec un minimum de parallélisme. Le parallélisme se situe au niveau du calcul de chaque pixel de sortie où toutes les multiplications entre les pixels de l'image et celles du masque sont effectuées en parallèle, suivie de la réduction. Cette version est testée sur différentes tailles de noyau de convolution pour évaluer les performances. De ce fait l'utilisation des ressources dépendra de la taille du noyau choisie car elle détermine le nombre de multiplication et d'accumulation pour chaque pixel de sortie. Sachant que sur FPGA, une multiplication flottante utilise un DSP alors la quantité totale de ressources utilisées est fortement impactée.

3.2 Déroulage de boucle

Nous pouvons ensuite exprimer plus de parallélisme en calculant plusieurs pixels de l'image de sortie en parallèle. Ce parallélisme est mise en œuvre à travers le déroulage de boucle qui consiste à répliquer le corps d'une boucle pour une exécution concurrente. Si le nombre de DSP le permet, plusieurs pixels de sortie peuvent être calculés en parallèle pour augmenter le débit du pipeline. Le seul inconvénient, dans ce cas, est la surcharge du bus de la mémoire globale du FPGA ce qui entraînera des blocages (pipeline stalls) dans l'exécution du pipeline. Dans ce cas, le déroulage est appliqué aux boucle sur les pixels de l'image afin d'en calculer plusieurs simultanément.

3.3 Utilisation de la mémoire locale

Pour alléger la charge du bus mémoire avec le degré de parallélisme, nous utilisons la mémoire locale du FPGA, dont la latence d'accès est très faible comparée à la mémoire globale, pour stocker les données. Avec une telle stratégie, nous réduisons le nombre d'accès en mémoire externe car nous chargeons, à la fois, en mémoire locale les données nécessaires au calcul d'une ligne de pixels de l'image de sortie. Le chargement de toutes ces données en mémoire

globale se fera alors en mode *burst* aligné car les données requises sont contiguës en mémoire.

En appliquant ces différentes optimisations en oneAPI et OpenCL, nous pourrons ainsi évaluer l'efficacité de ces deux outils sur la même gamme de FPGA.

4 Résultats et discussions

Dans cette section nous présentons les résultats de nos différentes implémentations sur FPGA en utilisant OpenCL et oneAPI. Dans nos expériences l'image *cameraman* de taille 512×512 pixels a été utilisée. Comme cartes accélératrices, le PAC Stratix 10 d'Intel et le DE10-Pro équipé d'un Stratix 10 ont été utilisé pour accélérer cet opérateur de convolution en effectuant une comparaison entre Intel oneAPI et Intel OpenCL SDK. Dans un souci de reproductibilité, le code source de ce projet est disponible en ligne [1].

4.1 Comparaison des performances

La figure 2 montre les temps d'exécution des différentes versions oneAPI et OpenCL sur des 5 tailles de noyaux de convolution différents.

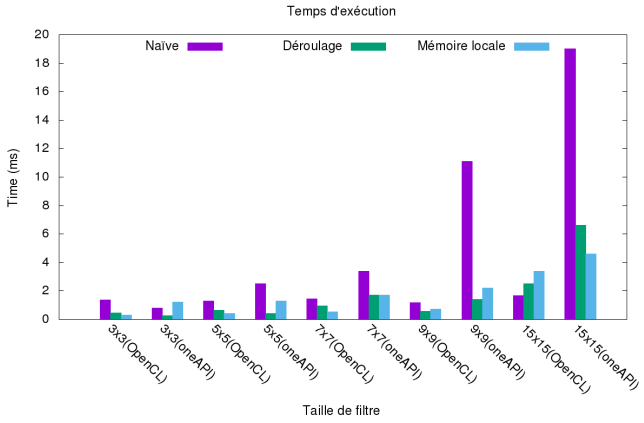

FIGURE 2 – Temps d'exécution de la convolution 2D sur FPGA

Les optimisations ont permis d'améliorer les performances de calcul par rapport aux versions naïves. Les versions naïves souffraient de blocage de pipeline dû à la forte sollicitation de la mémoire globale pour des accès mémoire non-alignés. En appliquant le déroulage de boucle sur plusieurs pixels la bande passante reste surchargée mais les performances sont meilleures que les versions naïves dû au degré de parallélisme. Ainsi, l'algorithme est rapidement caractérisé de *memory-bound* (limité par la mémoire). L'utilisation de la mémoire locale sur la puce FPGA a permis de soulager la pression sur le bus mémoire en surmontant le goulot d'étranglement de la mémoire globale. Ce soulagement de la bande passante est rendu possible grâce au préchargement et l'accès aux données de manière contiguë. En analysant l'outil de profilage d'Intel, nous constatons que le pourcentage de blocage du pipeline dû aux accès mémoire a été considérablement réduit. Par contre le taux d'occupation de nos unités de calcul reste assez faible ce qui montre qu'il existe encore des possibilités d'optimisation supplémentaire. En effet, les données pré-chargées en mémoire locale ne sont réutilisées que par les pixels d'une même ligne alors qu'une partie de ces données est utilisée par les lignes de pixels suivantes. Nous effectuons plus d'accès mémoire qu'il n'en faut ce qui pénalise les performances de certaines versions utilisant la mémoire locale notamment pour les masques de taille grande (7×7, 9×9, 15×15). Une amélioration de l'intensité arithmétique s'impose pour une utilisation intensive des unités de calcul du FPGA. Pour ce faire, nous devons réutiliser les données autant que possible dans la mémoire locale.

	3×3	5×5	7×7	9×9	15×15
Naïve	0.6	1.9	2.4	9.5	11.4
Déroulage de boucle	0.6	0.6	1.7	2.4	4
Mémoire locale	4	3.1	3.3	3	1.4

TABLE 1 – Facteur d'accélération d'OpenCL par rapport à oneAPI

Nous constatons une meilleure performance pour l'outil OpenCL dans quasiment tous les cas de figure pour l'opérateur de convolution comme illustré dans la table 1. Cet avantage de performance de l'OpenCL réside dans la différence de profondeur du pipeline généré. En effet, nous avons remarqué une différence importante dans les latences d'accès en mémoire globale de ces deux outils (201 versus 818 cycles). Cette latence mémoire se répercute sur le nombre d'étages du pipeline notamment dans les blocs où l'accès à la mémoire globale est requis. Pour un pipeline idéal, ceci a peu d'impact sur la performance dans la mesure où le pipeline est capable de produire un résultat à chaque cycle d'horloge. Peu importe la profondeur du pipeline, une fois rempli il est capable de sortir un résultat à chaque coups d'horloge (ou à intervalles réguliers) sachant que l'on travaille sur des données de très grande taille. Par contre, en cas de blocage de calcul pour faute des données (pourcentage de stall élevé) tous les étages du pipeline seront impactés et ce retard se propagera tout le long de la profondeur ce qui empêchera de produire un résultat à chaque coup d'horloge. OneAPI reste moins avantageux à ce niveau avec une latence d'accès en mémoire globale 4× plus élevée que celle de l'OpenCL et cette latence aura une répercussion sur le pipeline généré. Dans tous les cas, l'accès à la mémoire globale doit être limité pour garantir de meilleurs performances et pour cela l'utilisation de la mémoire locale (faible latence) devra être privilégier.

Pour que les FPGAs s'imposent comme candidat pour les applications de calcul intensif, les vendeurs doivent conti-

1. https ://github.com/ddiakite/test_convolution.git

nuer à apporter des améliorations aux outils de synthèse haut niveau. En particulier, pour oneAPI sur le PAC Stratix 10 d'Intel il y a un besoin d'améliorer les communications entre le noyau de calcul et la mémoire DDR.

4.2 Comparaison de métriques

Nous présentons dans la table 2 la consommation des ressources matérielles ainsi que la fréquence maximale du pipeline pour un masque de taille 9x9 générer par les deux outils. Nous remarquons que les outils gèrent les ressources FPGA de la même manière avec une légère différences dû entre autre à la version du BSP (Board Support Package). En revanche, le grand écart de consommation de BRAM s'explique par la réplication de la mémoire locale pour supporter plusieurs accès concurrents. En terme de fréquence

Ressource	DSP	BRAM	LUT	Fmax (MHz)
OneAPI	332 (6%)	1126 (10%)	224454	340
OpenCL	332 (6%)	2077 (18%)	134756	309

TABLE 2 – Ressources consommées

de fonctionnement, oneAPI arrive à atteindre des fréquences plus élevées pour la même configuration du noyau de calcul par rapport à OpenCL.

4.3 Temps de développement

Les deux outils offrent tous la possibilité d'utiliser un même langage de programmation pour cibler différentes plate-formes. Toutefois, oneAPI présente une simplicité en plus qui consiste à utiliser un seul et même fichier pour exprimer le code de la plate-forme hôte et celui du kernel. Le framework oneAPI permet de réduire considérablement l'effort de programmation car les outils prennent en charge le plus grand du travail du côté de l'hôte. Le nombre de ligne de code pour une application oneAPI étant nettement inférieur à celui d'une application OpenCL, le temps de développement en oneAPI reste plus avantageux qu'en OpenCL ce qui entraîne une plus grande productivité.

5 Conclusion

Cet article traite de l'accélération de l'opérateur de convolution 2D sur FPGA avec des langages de haut niveau. Plusieurs implémentation ont été mise en œuvre pour évaluer les outils de synthèse OpenCL et oneAPI d'Intel. Ces outils permettent d'avoir une implémentation sur FPGA avec un temps de développement réduit, mais un design performant nécessite une analyse approfondie de l'application afin d'appliquer des optimisations conséquentes. Nous avons effectué une comparaison de ces deux outils HLS d'Intel sur l'opérateur de convolution et nous constatons que OpenCL offre le meilleur compromis entre temps de développement et performance. Toutefois, oneAPI gagne de plus en plus en maturité et pourra s'imposer comme alternative à l'OpenCL.

Enfin, nos travaux actuel et futur concernent une optimisation plus avancée de cet opérateur en réutilisant intensivement les données pré-chargées en mémoire pour améliorer l'intensité arithmétique de l'opérateur. Nous comptons inclure dans cet étude d'autres applications de calcul intensif (e.g., tomographie) afin d'avoir une comparaison plus complète de ces outils de synthèse. En outre, nous effectuerons une comparaison avec les travaux basées sur FPGA à travers les outils HLS, CPU et GPU rapportées dans la littérature.

Remerciement

Nous tenons à remercier Intel pour nous avoir accordé l'accès à son DevCloud pour utiliser le PAC Stratix 10 avec le framework oneAPI.

Références

[1] W. Burger and M. J. Burge, "Digital image processing : an algorithmic introduction using java," 2016.

[2] J. Idier, *Bayesian approach to inverse problems*. John Wiley & Sons, 2013.

[3] J. Fowers, G. Brown, J. Wernsing, and G. Stitt, "A performance and energy comparison of convolution on gpus, fpgas, and multicore processors," *ACM Trans on Architecture and Code Optimization (TACO)*, 2013.

[4] Z. Jin and H. Finkel, "Exploration of OpenCL 2d convolution kernels on intel FPGA, CPU, and GPU platforms," in *2019 IEEE International Conference on Big Data*, 2019, pp. 4460–4465.

[5] S. Winograd, *Arithmetic complexity of computations*. Siam, 1980, vol. 33.

[6] E. O. Brigham, *The fast Fourier transform and its applications*. Prentice-Hall, Inc., 1988.

[7] M. Seznec, N. Gac, F. Orieux, and A. S. Naik, "A new convolutions algorithm to leverage tensor cores," GPU Technology Conference (GTC), May 2020, poster.

[8] A. Aguilar-González, M. Arias-Estrada, M. Pérez-Patricio, and J.-L. Camas-Anzueto, "An fpga 2d-convolution unit based on the caph language," *Journal of Real-Time Image Processing*, 2015.

[9] Intel oneAPI programming guide. [Online]. Available : https ://www.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top.html

[10] R. Keryell, M. Rovatsou, and L. Howes, "Sycl specification generic heterogeneous computing for modern c++," p. 274, 2020.

[11] H. Ström, "A parallel fpga implementation of image convolution," 2016.