Origami : L'art du pliage appliqué à l'accélération des CNNs sur FPGA

Léo PRADELS Rémi GARCIA Silviu-Ioan FILIP Olivier SENTIEYS Daniel CHILLET
Université de Rennes, IRISA, Inria

Résumé – Les réseaux neuronaux convolutifs sont couramment utilisés dans les systèmes embarqués pour des applications temps réel. Cependant, les ressources matérielles limitées et les exigences de performance strictes en termes de débit nécessitent des stratégies d'optimisation efficaces. Cet article présente Origami, un outil combinant une architecture pipelinée et une méthode de pliage optimisée par programmation linéaire en nombres entiers. Cette approche améliore les performances tout en réduisant l'utilisation des ressources FPGA, avec des gains allant jusqu'à 43% de débit et 30% de ressources logiques en moins par rapport à des solutions comme FINN. Origami constitue ainsi une solution efficace pour l'accélération de CNNs en environnement contraint.

Abstract – Convolutional neural networks are commonly used in embedded systems for real-time applications. However, limited hardware resources and strict performance requirements in terms of throughput call for efficient optimization strategies. In this work, we present Origami, a tool combining a pipelined architecture and a folding method optimized by integer linear programming. This approach improves performance while reducing the use of FPGA resources, with gains of up to 43% in throughput and 30% in logic resources compared with solutions such as FINN. Origami is therefore an effective solution for accelerating CNNs in constrained environments.

1 Introduction

Les réseaux neuronaux convolutifs (CNN) correspondent à l'une des méthodes les plus avancées pour de nombreuses tâches de vision par ordinateur. Par exemple, la Super Résolution (SR) utilise des CNN [4, 6]. Le déploiement de ces réseaux sur des systèmes embarqués est complexe en raison de leurs exigences en matière de calcul, générant un coût élevé en termes de consommation énergétique et d'utilisation mémoire.

L'architecture de l'accélérateur joue un rôle primordial sur les performances du réseau pendant l'inférence, et il en existe deux grandes familles. Les architectures *séquentielles* [2] utilisent un bloc matériel général et flexible (*e.g.*, optimisé pour les opérations générales de multiplications matricielles) qui exécute chaque couche l'une après l'autre. Les architectures *pipelinées* [1, 4] calculent en parallèle (de manière pipelinée) toutes les couches sur un FPGA cible. Dans ce travail, nous utilisons une architecture pipelinée. Ce type d'architecture permet de réduire le trafic hors puce [8], ce qui est notamment une nécessité dans le cas des applications SR [4] temps réel.

Chaque couche est elle-même implémentée en parallèle, à l'aide de « tuiles » qui recouvrent la totalité de la couche. La performance d'un accélérateur dépend du choix de la taille de ces tuiles [9, 10, 12]. L'exploration de l'espace de conception (DSE, pour *Design Space Exploration*) de l'accélérateur consiste en la recherche des tailles de tuiles les plus intéressantes pour chacune des couches. Les tailles des tuiles, appelées pliages, sont déterminées en fonction de plusieurs critères, tels que la taille de l'image d'entrée ou l'architecture du CNN.

Les différents pliages possibles définissent un vaste espace de conception à explorer et mettent en évidence un compromis entre les performances et les ressources. Au fur et à mesure que les modèles CNN et les dispositifs FPGA deviennent plus grands, l'espace de conception qui doit être exploré pour l'identification des conceptions dîtes Pareto-optimales augmente, ce qui nécessite l'utilisation de méthodes automatisées pour l'exploration. Une approche classique, utilisée pour déterminer le pliage, est la Recherche d'Architecture Neuronale (NAS, pour *Neural Architecture Search*) [7]. Cette approche est très coûteuse en temps de calcul en raison de l'exploration d'architectures en plus du simple pliage. Nous ne l'avons donc pas utilisée pour ce travail.

Plusieurs outils, comme FINN [1], fpgaConvNet [11], HLS4ML [3] ou SAMO [9], existent et permettent de réaliser automatiquement la DSE pour des temps de calcul raisonnable. Par exemple, FINN [1] commence par une allocation minimale de ressources, puis déplie progressivement les couches les plus lentes selon des règles spécifiques, afin d'optimiser le pliage du réseau. SAMO [9] compare plusieurs stratégies que sont le recuit simulé (une métaheuristique), une méthode gloutonne et un algorithme force brute. À l'exception de la force brute, dont les temps de calcul sont rédhibitoires, ces méthodes ne garantissent pas l'optimalité du pliage proposé.

2 Origami

Dans ce travail, nous présentons Origami, un nouvel outil intégrant un accélérateur pipeliné pour le traitement d'image et une méthode d'optimisation du pliage par Programmation Linéaire en Nombres Entiers (PLNE).

Conception de l'Accélérateur Pipeliné Origami

Les architectures d'accélérateurs existantes, comme celles proposées par FINN [1] ciblent principalement les CNNs utilisés pour la classification, comme ResNet [5]. Ces CNNs n'ont que peu de données intermédiaires (FM pour *Features Maps*)) entre les couches. D'autres tâches, comme la SR, génèrent des FMs bien plus volumineuses. Par exemple, VDSR 20 [6] a $64 \times 19 \times 800 \times 800$ (pour une image d'entrée 800×800) valeurs pour stocker toutes ses FMs, soit 778.24 Mo pour des FMs 8-bit. Des FPGAs comme sur la ZCU104 ne disposent que de 5 Mo de mémoire interne et les architectures proposées par FINN ne peuvent donc pas être utilisées sur celui-ci.

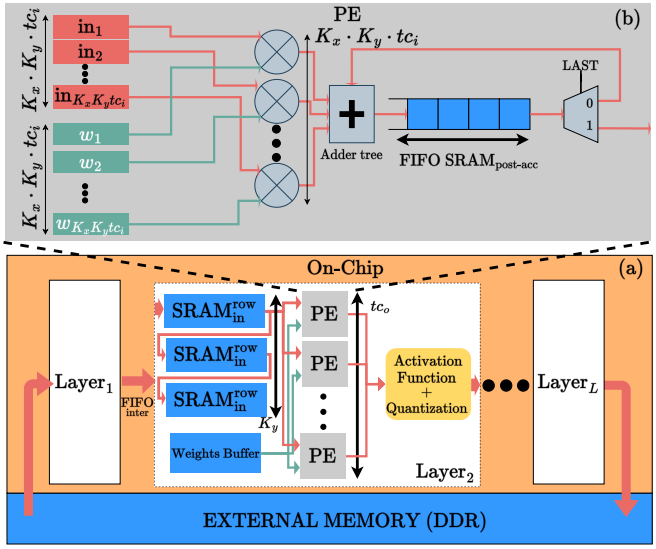

FIGURE 1 : Architecture globale de l'accélérateur, avec un pliage de taille tc_i la taille des tuiles sur les canaux d'entrée et tc_o le nombre de PEs. Les parties bleues indiquent les composants de la mémoire, tandis que les parties grises représentent les PEs.

Pour pallier à cela, nous proposons de réduire le volume de FMs au sein du FPGA via un décalage ligne-colonne des entrées comme présenté dans [4] (Fig. 1(a), en bleu, SRAM$^{\text{row}}_{\text{in}}$). Ce décalage ligne-colonne est effectué de la façon suivante. Dans les SRAMs, nous stockons successivement autant de lignes que la taille du noyau de convolution de la couche. Puis chaque SRAM alimente un registre à décalage correspondant également à la taille du noyau. Cela permet alors d'envoyer simultanément toutes les entrées aux unités de calcul.

Nous exploitons également les URAMs disponibles sur la ZCU104 afin d'améliorer les performances, contrairement à FINN, qui ne les utilise pas dans sa conception. L'architecture globale de notre accélérateur est illustrée en Fig. 1(a), où l'unique accès à la DDR concerne la lecture/écriture des entrées et sorties du réseau. En bleu, dans la Fig. 1(a), nous indiquons les SRAM$^{\text{row}}_{\text{in}}$ correspondant aux décalages ligne-colonne.

Chaque couche intègre tc_o éléments de calcul (PE pour *Processing Element*), détaillés dans la Fig. 1(b), et utilise $K_x \times K_y \times tc_i$ DSPs, où K_x, K_y sont les dimensions des noyaux, et tc_i le facteur de pliage des canaux d'entrée. Plus les facteurs de pliage sont élevés, plus la latence de la couche diminue grâce à un déroulement plus important des calculs. En contrepartie, la consommation en DSP augmente. Il s'agit donc de trouver un compromis entre latence et consommation en DSP.

Les poids sont stockés dans la mémoire interne du FPGA, au sein des *Weight Buffers* (Fig. 1(a)), sur des URAMs, avec un noyau de convolution par ligne. En jaune, on distingue l'unité chargée de calculer la fonction d'activation après l'accumulation, puis d'appliquer la quantification avant de transmettre les données à la FIFO pour la couche suivante.

Les FMs sont stockées dans des mémoires tampons, FIFO, entre les couches implémentées avec des mémoires internes du FPGA (BRAMs ou URAMs). La profondeur de ces FIFOs est déterminée par l'écart de latence entre couches pour transmettre un pixel valide. Un débit similaire entre couches est donc crucial pour limiter l'usage de la SRAM (BRAM, URAM). À la couche ℓ, les FIFOs ont une largeur de $tc_i^{\ell+1}$.

Exploration de l'Espace de Conception via la PLNE

Notre objectif est de proposer une modélisation mathématique du problème de pliage. Notre optimisation se concentre sur l'usage de la SRAM et des DSPs, en excluant les ressources logiques (LUTs et FFs), dont l'estimation est souvent imprécise [9] et qui ne constituent pas un facteur limitant dans notre cas. Pour cela, une description complète et précise du problème que nous cherchons à traiter est nécessaire. Les objectifs du pliage sont donc les suivants :

- Minimiser la latence en réduisant le temps nécessaire pour qu'une entrée unique traverse l'ensemble du réseau, garantissant ainsi des temps de réponse rapides ;
- Maximiser le débit du réseau et donc de chaque couche ;
- Minimiser la différence de débit et de latence entre chaque couche en dessous d'un seuil donné ;
- Contraindre l'utilisation de ressources : veiller à ce que les configurations proposées ne dépassent pas les ressources disponibles du FPGA cible.

On peut noter que le débit de l'accélérateur est limité par la couche la plus lente, d'où l'importance de maximiser le débit de chaque couche. La latence est également directement impactée par la couche la plus lente : d'une certaine façon, toutes les couches sont alignées sur cette latence maximale via l'ajout de FIFO entre chacune des couches. Réduire ces écarts diminue ainsi la profondeur des FIFOs, et donc la consommation de mémoire interne. Nous pourrions effectuer une exploration exhaustive de l'espace de conception en testant tous les pliages possibles afin de sélectionner la configuration optimale, il s'agit d'une méthode force brute. Malheureusement, le nombre de configurations croît de manière exponentielle et cette méthode n'est pas applicable en pratique.

Nous rapportons, dans la table 1, quelques valeurs obtenues par l'application force brute sur des réseaux VDSR [6] et le réseau ResNet-20 [5]. Cette expérience a été menée pour une carte Xilinx ZCU104, donc 1728 DSPs. Pour les réseaux VDSR L, le nombre de couches du réseau correspond à la valeur de L et les entrées sont de taille $3 \times 200 \times 200$ visant un débit de 30 Image Par Seconde (IPS). Pour le réseau ResNet-20, les entrées sont de taille $3 \times 32 \times 32$.

La table présente le nombre total de configurations possibles, $N^{\text{total}}_{\text{pliage}}$, par réseau et le nombre de configurations valides, $N^{\text{valide}}_{\text{pliage}}$, respectant les contraintes d'entrée. Le temps de d'exécution total de l'approche par force brute est rapporté dans la dernière colonne pour chaque cas. Dans certaines situations, ce temps correspond à une estimation en raison du temps de calcul trop élevé.

Le temps de recherche par cette méthode croît de façon exponentielle avec la profondeur du réseau, en raison de la croissance de $N^{\text{total}}_{\text{pliage}}$. Par exemple, pour VDSR 10, pour une carte ZCU104, 67.88×10^{31} pliages sont à tester. Aucun de ces pliages n'est valide et, pour le vérifier, nous estimons le temps de calcul à 1.86×10^{26} secondes, ce qui correspond à environ 5.9×10^{18} ans. Des limitations similaires sont observées pour ResNet-20. Ainsi, pour des réseaux profonds, une méthode plus efficace devient indispensable afin de trouver une configuration optimale dans un temps raisonnable. Notre premier choix a été de réduire l'espace de recherche en intégrant des contraintes supplémentaires. L'algorithme par force brute initial devient donc un algorithme force brute « sur-contraint ».

TABLE 1 : Temps de DSE pour une méthode force brute pour différents réseaux. L'objectif de débit est de 30 IPS, pour une carte ZCU104. N_{pliage} correspond au nombre de pliages.

Réseau	$N_{\text{pliage}}^{\text{total}}$	$N_{\text{pliage}}^{\text{valide}}$	Temps DSE (s)
VDSR 4	12.54×10^3	106	0.008
VDSR 5	61.47×10^4	324	0.2
VDSR 6	30.12×10^6	81	6.03
VDSR 7	1.48×10^9	62	192.65 (3min12s)
VDSR 8	72.31×10^9	37	7168.89 (1h59min29s)
VDSR 9	8.507×10^{15}	$20^\dagger$	$2.33 \times 10^{9\dagger}$ (74a)
VDSR 10	67.88×10^{31}	$0^\dagger$	$1.86 \times 10^{26\dagger}$ (5.90×10^{18}a)
ResNet-20	5.10×10^{28}	$16900^\dagger$	$1.40 \times 10^{23\dagger}$ (4.43×10^{15}a)

†indique une estimation.

Autrement dit, nous abandonnons l'assurance d'optimalité pour réduire les temps de calcul. Nous ne présentons pas les détails de ces contraintes supplémentaires dans ce résumé.

Afin de conserver la possibilité d'obtenir une solution optimale, nous avons fait le choix d'utiliser la modélisation mathématique, par la programmation linéaire en nombre entier (PLNE) en particulier. Pour cela, nous devons transposer mathématiquement les règles décrites précédemment :

1. Latence de la couche. La latence de chaque couche, $\mathcal{L}_\ell$, doit se situer dans la plage cible, c'est-à-dire être inférieure à la latence maximale de chaque couche : $\mathcal{L}_\ell \leq \mathcal{L}^{\max}$. Cela signifie donc que

$$\mathcal{L}^{\max} = \max_{\ell \in [\![1;L]\!]} \mathcal{L}_\ell. \tag{1}$$

2. Consommation des ressources d'une couche. La consommation totale de DSP sur l'ensemble des couches N_{DSP}, calculée en additionnant l'utilisation de DSP de chaque couche $N_{\text{DSP}} = \sum_{\ell=1}^{L} N_{\text{DSP}}^\ell$, ne doit pas dépasser la limite de DSPs disponible sur le FPGA, $N_{\text{DSP}} \leq M_{\text{DSP}}$.

3. Un seul pliage par couche. Parmi toutes les configurations possibles pour une couche, une seule configuration de pliage doit être choisie pour chaque couche.

4. Correspondance entre les plis. Le pliage du canal de sortie de chaque couche tc_o^ℓ doit être un diviseur du pliage du canal d'entrée de la couche suivante, $tc_i^{\ell+1}$. Cela peut s'exprimer comme suit :

$$tc_i^{\ell+1} \equiv 0 \pmod{tc_o^\ell}. \tag{2}$$

5. Diviseur du canal. Les pliages tc_i et tc_o doivent être des diviseurs du canal d'entrée et de sortie, respectivement. Formellement, cela peut s'écrire :

$$C_i \equiv 0 \pmod{tc_i}, \tag{3}$$
$$C_o \equiv 0 \pmod{tc_o}. \tag{4}$$

Nos principaux objectifs dans la recherche de pliage sont de minimiser la latence maximale des couches $\mathcal{L}^{\max}$, comme définie dans l'équation (1), et de réduire la variation de latence entre les couches $\mathcal{L}^{\text{sum}}$. Cette variation de latence peut être encodée comme la différence des temps de latence :

$$\mathcal{L}^{\text{sum}} = \sum_{\ell=1}^{L} \mathcal{L}^{\max} - \mathcal{L}_\ell. \tag{5}$$

Minimiser $\mathcal{L}^{\text{sum}}$ permet de réduire la surcharge de SRAM dans les FIFOs inter-couches.

Mathématiquement, cela signifie que nous pouvons écrire que nos objectifs sont de :

1. Minimiser la latence maximale :

$$\min \mathcal{L}^{\max}; \tag{6}$$

2. Minimiser la différence de temps de latence :

$$\min \mathcal{L}^{\text{sum}}. \tag{7}$$

La PLNE nécessite de décrire le modèle mathématique sous formes d'un ou plusieurs objectifs avec des contraintes linéaires. Toutes les règles de notre modèle peuvent être exprimées sous forme linéaire et nous faisons donc le choix de les exprimer ainsi afin d'utiliser l'efficacité des solveurs linéaires.

3 Expériences

Nous évaluons notre méthode sur plusieurs CNNs afin de démontrer son efficacité. Les résultats incluent le temps de recherche, les IPS et la somme des latences inter-couches $\mathcal{L}^{\text{sum}}$ pour la configuration optimale. Le temps de recherche comprend la génération du modèle PLNE, implémenté en Julia et résolu par le solveur Gurobi 12. Nous proposons également une comparaison des implémentations matérielles de notre accélérateur, développé en High-Level Synthesis (HLS) à l'aide de Vitis HLS 2022.2, avec FINN 2022.1 en ciblant la carte de développement Xilinx ZCU104.

Les résultats d'optimisation sont présentés dans la table 2. Comparée à l'approche exhaustive, notre méthode PLNE réduit significativement les temps de recherche, quel que soit le nombre de couches. Dans certains cas (*e.g.*, VDSR 15 et 20), aucune configuration valide n'est trouvée, ce qui indique que le déploiement temps réel n'est pas réalisable. Le solveur arrive rapidement à cette conclusion, contrairement à l'approche exhaustive qui doit explorer toutes les options avant de conclure que le problème n'a pas de solution. Globalement, cette méthode permet une exploration efficace de l'espace de conception avec des temps de calcul réduits.

Nous avons également constaté une grande différence entre ResNet-20 et VDSR dans les résultats d'optimisation. La taille réduite des images d'entrée (32×32) pour ResNet permet notamment un débit nettement plus élevé pour que VDSR. En revanche, la disparité entre les couches de ResNet-20, contrairement à VDSR dont les couches n'ont que peu de variabilité, génère un besoin important de FIFOs entre les couches. Cela se traduit par une $\mathcal{L}^{\text{sum}}$ nettement plus élevé que pour les VDSR.

La table 3 compare notre outil à FINN [1], une solution de référence pour l'accélération de CNNs sur FPGA. Notre méthode offre des performances similaires, voire supérieures, en termes de débit et d'utilisation des ressources FPGA. Les différences observées dans la consommation de ressources proviennent des variations de pliage et de l'implémentation des couches. FINN ne tire pas parti des URAMs, mémoires plus grandes que les BRAMs (4096×72bits contre 1024×36bits), contrairement à notre approche, ce qui explique une partie de l'écart sur les BRAMs. Par ailleurs, notre pliage est optimisé pour réduire les variations de latence entre couches, limitant ainsi l'utilisation de FIFO intermédiaires, un aspect non pris en compte dans FINN. Sur ResNet-20, nous atteignons 811

TABLE 2 : Comparaison du temps de DSE pour différents réseaux. L'objectif de débit est de 30 IPS, pour une carte ZCU104 avec une fréquence d'horloge de 200 MHz. Les valeurs avec X correspondent aux cas où le pliage est infaisable.

Methode	Réseau	IPS	$\mathcal{L}^{\text{sum}}$ (cycles)	Temps DSE (s)
	VDSR 5	88.51	2304	0.129
	VDSR 10	31.45	6913	0.127
PLNE	VDSR 15	X	X	0.00023
	VDSR 20	X	X	0.0003
	ResNet-20	811.03	40329	0.238

TABLE 3 : Comparaison des IPS et des ressources entre FINN et notre méthode (PLNE) sur VDSR et ResNet-20. La taille de l'image est $S \times S$. Les valeurs en rouge sont inférieures à l'objectif d'IPS. Les ressources FPGA sont données pourcentage pour une carte Xilinx ZCU104.

Méthode	Réseau L	S	IPS	LUT	DSP	BRAM	URAM
FINN	VDSR 5	200	62	27.3	58.3	35.9	0
PLNE			88.5	19.1	87.5	9.0	100
FINN		400	**15.4**	37.0	58.9	36.2	0
PLNE			**23.5**	12.8	92.2	12.3	15.6
FINN	VDSR 10	200	31.4	45.3	76.0	46.8	0
PLNE			31.5	14.6	76.0	26.8	53.1
FINN		400	**7.7**	66.9	76.0	46.8	0
PLNE			**7.7**	14.0	89.1	20.7	37.5
FINN	ResNet-20	32	42.4	32.0	1.5	19.3	0
PLNE			811	5.8	71.9	33.8	62.5

IPS, contre 42,4 pour FINN. Cet écart est possible grâce à l'optimisation plus efficace de l'utilisation des DSPs par notre approche. Cet avantage entraîne également une diminution de l'utilisation de LUTs. On observe notamment que, pour un débit similaire, comme dans le cas de VDSR 10, notre méthode consomme moins de LUTs que FINN.

4 Conclusion & Travaux Futurs

Dans ce travail, nous présentons Origami, un nouvel outil permettant d'accélérer des CNNs sur FPGA à l'aide d'un accélérateur pipeliné. Le pliage est optimisé à l'aide d'une méthode de PLNE. Grâce à la PLNE, nous prenons en compte l'ensemble des contraintes de l'architecture et nous obtenons des solutions avec preuve d'optimalité. Ainsi, Origami permet d'obtenir des implémentations moins coûteuses que les outils de l'état de l'art comme FINN. Dans de futurs travaux, nous prévoyons une comparaison approfondie avec d'autres méthodes, notamment celle proposées dans SAMO [9]. La flexibilité apportée par la PLNE pourra également être utilisée pour intégrer l'optimisation d'autres critères, notamment de la consommation énergétique.

L'une des limitations des méthodes existantes et de notre approche vient de la taille des réseaux et des données. Pour des réseaux avec un nombre important de couches ou des images d'entrée plus grandes, il devient difficile de représenter le réseau sur des FPGAs pour un traitement en temps réel. Cette limitation est inhérente aux accélérateurs de flux de données ; l'augmentation de la taille de l'image d'entrée ou de la profondeur du réseau se traduit par un nombre plus élevé d'opérations de multiplication accumulation, ce qui nécessite des ressources de calcul supplémentaires pour maintenir le débit cible. Pour y remédier, une technique de compression comme l'élagage peut aider à diminuer l'utilisation de ressources.

Références

[1] Michaela BLOTT, Thomas B PREUSSER, Nicholas J FRASER, Giulio GAMBARDELLA, Kenneth O'BRIEN, Yaman UMUROGLU, Miriam LEESER et Kees VISSERS : FINN-R : An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. *ACM TRETS*, 2018.

[2] Yu-Hsin CHEN, Tushar KRISHNA, Joel EMER et Vivienne SZE : Eyeriss : An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *In IEEE ISSCC*, 2016.

[3] Farah FAHIM, Benjamin HAWKS, Christian HERWIG, James HIRSCHAUER, Sergo JINDARIANI, Nhan TRAN, Luca P CARLONI, Giuseppe DI GUGLIELMO, Philip HARRIS, Jeffrey KRUPA *et al.* : hls4ml : An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices. *arXiv*, 2021.

[4] Koen GOETSCHALCKX et Marian VERHELST : Breaking high-resolution CNN bandwidth barriers with enhanced depth-first execution. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

[5] Kaiming HE, Xiangyu ZHANG, Shaoqing REN et Jian SUN : Deep Residual Learning for Image Recognition. *In 2016 IEEE CVPR*. IEEE, 2016.

[6] Jiwon KIM, Jung Kwon LEE et Kyoung Mu LEE : Accurate Image Super-Resolution Using Very Deep Convolutional Networks. *In IEEE CVPR*, 2016.

[7] Yuhong LI, Cong HAO, Xiaofan ZHANG, Xinheng LIU, Yao CHEN, Jinjun XIONG, Wen-mei HWU et Deming CHEN : Edd : Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions. *In IEEE DAC*, 2020.

[8] Linyan MEI, Koen GOETSCHALCKX, Arne SYMONS et Marian VERHELST : DeFiNES : Enabling Fast Exploration of the Depth-first Scheduling Space for DNN Accelerators through Analytical Modeling. *In IEEE HPCA*, 2023.

[9] Alexander MONTGOMERIE-CORCORAN, Zhewen YU et Christos-Savvas BOUGANIS : SAMO : Optimised Mapping of Convolutional Neural Networks to Streaming Architectures. *In IEEE FPL*, 2022.

[10] Angshuman PARASHAR, Priyanka RAINA, Yakun Sophia SHAO, Yu-Hsin CHEN, Victor A YING, Anurag MUKKARA, Rangharajan VENKATESAN, Brucek KHAILANY, Stephen W KECKLER et Joel EMER : Timeloop : A Systematic Approach to DNN Accelerator Evaluation. *In IEEE ISPASS*, 2019.

[11] Stylianos I. VENIERIS et Christos-Savvas BOUGANIS : fpgaConvNet : Mapping Regular and Irregular Convolutional Neural Networks on FPGAs. *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[12] Xuan YANG, Mingyu GAO, Qiaoyi LIU, Jeff SETTER, Jing PU, Ankita NAYAK, Steven BELL, Kaidi CAO, Heonjae HA, Priyanka RAINA, Christos KOZYRAKIS et Mark HOROWITZ : Interstellar : Using Halide's Scheduling Language to Analyze DNN Accelerators. *In ASPLOS*, 2020.