Projet

Analyse énergétique d'un processeur vectoriel pour des calculs de DNN
Rapport final

ELE6307 - Machines neuronales : architectures et applications

Hiver 2022
Département de génie électrique
École Polytechnique de Montréal

Dernière mise à jour: 21 avril 2022

Yoan **Fournier** 1958736

Victor **Gaudreau-Blouin** 1958297

1 Introduction

Dans les dernières années, l'utilisation de réseaux de neurones profonds (DNN) pour résoudre différentes tâches a beaucoup augmenté. Ces DNNs nécessitent des puissances de calculs considérables, mais à la fois nécessitent une bonne efficacité énergétique étant donné leur déploiement dans des appareils mobiles. Bien que les processeurs généralistes que l'on retrouve aujourd'hui ont augmenté rapidement en performance, les limitations du *Dennard Scaling* se font ressentir. Il est donc souhaitable de trouver une autre approche pour accélérer de façon efficace les calculs dans les DNNs. Une manière intéressante pour l'accélération des calculs pour des DNNs est l'utilisation de processeurs vectoriels plutôt que des processeurs scalaires standards. Ces processeurs utilisent un jeu d'instruction SIMD (*Single instruction, Multiple Data*) plutôt que SISD (*Single instruction, Single data*). Ainsi une instruction SIMD peut performer la même opération sur plusieurs données plutôt qu'une seule, ce qui permet de plus facilement augmenter le parallélisme des calculs.

Le présent rapport étudie l'efficacité énergétique d'un modèle simplifié du processeur vectoriel ARA sur différents problèmes en utilisant le simulateur *Timeloop-Accelergy*.

2 Méthodologie

L'architecture sur laquelle notre modèle se base est sur celle du coprocesseur vectoriel ARA [2]. Ce coprocesseur est une extension vectorielle du processeur scalaire RISCV CVA6. Le fonctionnement d'ARA se base sur le concept de *Lanes*. Les figures 1 et 2 présentent l'architecture d'ARA [1]. Le coprocesseur ARA possède un nombre d'allées, ou *Lanes*, qui sont capables de traiter des éléments vectoriels indépendamment. Le routage des données vers ces *Lanes* grandit évidemment en complexité avec le nombre de *Lanes*. Le *Sequencer* s'occupe de lire les instructions à exécuter et transmettre les informations correctes au *Vector Load and Store Unit* (VLSU), *Slide Unit* (SLDU) ou aux *Lanes*. Le VLSU est responsable de générer les bonnes adresses, amenant les données séquentiellement vers et depuis les *Lanes*. Le SLDU exécute des opérations qui n'ont pas d'indépendance entre les *Lanes*. Finalement, chaque *Lane* a son propre *sequencer* pour les instructions à exécuter sur son élément vectoriel. Le reste de la *Lane* peut être considéré comme un PE, avec sa mémoire locale, un étage de conversion et un étage d'exécution.

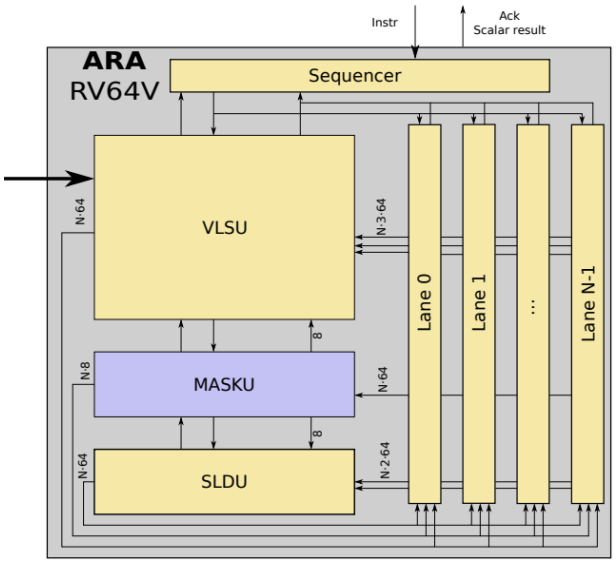

FIGURE 1 – Architecture globale

Afin de modéliser l'efficacité énergétique du coprocesseur vectoriel ARA, le simulateur Ti-

meloop a été utilisé. Ce simulateur utilise une structure hiérarchique pour modéliser une architecture. Une structure détaillant le problème à résoudre par l'architecture est aussi spécifiée. La commande *timeloop-mapper* permet ensuite de trouver automatiquement la correspondance idéale du problème sur l'architecture matérielle.

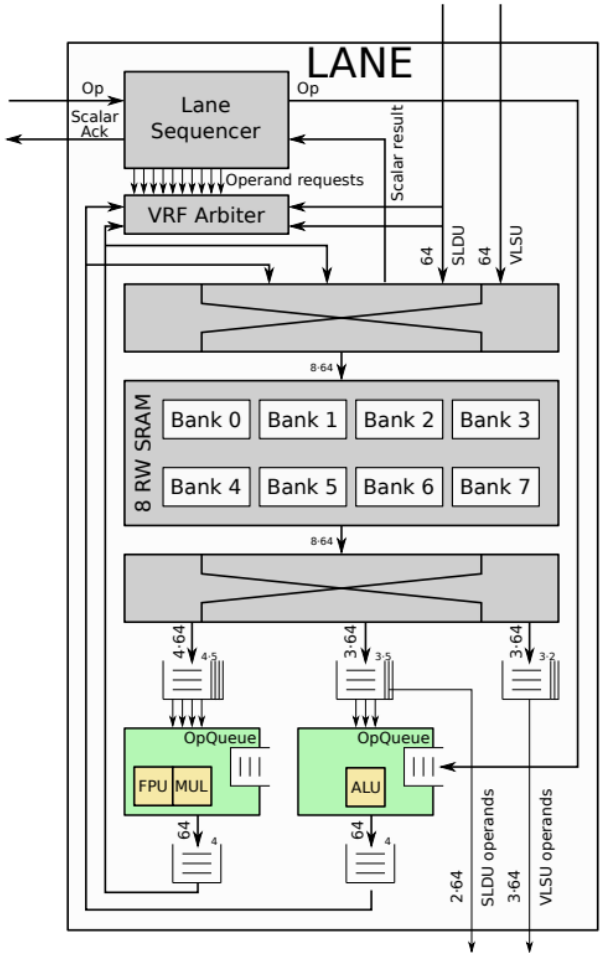

FIGURE 2 – Architecture *Lane*

L'architecture simplifiée d'ARA, comme mentionnée dans la proposition du projet, peut se modéliser en termes de *Lanes*. Chaque *Lanes* à sa plus simple expression consiste en une banque de 8 registres ainsi qu'une unité arithmétique capable d'effectuer des multiplications et des additions. Bien que sur ARA, l'unité d'addition est séparée de l'unité de multiplication, les deux unités peuvent fonctionner durant le même cycle sur des données indépendantes. Dans le cadre de la modélisation, on simplifie en utilisant la structure *intmac* fournie par l'outil Timeloop-Accelergy. Nous considérons que la différence causée par cette simplification est minimale. En effet, les opérations dans une Lane sont pipelinées et les opérations de multiplication et addition peuvent être exécutées en même temps. La seule différence principale est la latence causée par le pipelinage des opérations. Cependant, ce n'est pas une métrique de performance étudiée dans le cadre de ce projet.

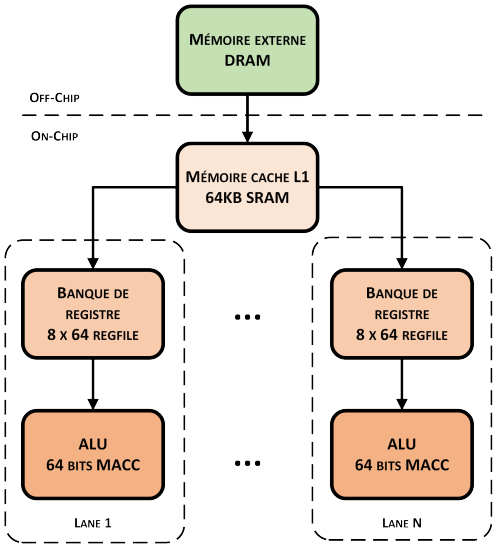

FIGURE 3 – Architecture de ARA sur Timeloop

L'architecture utilisée est présentée à la figure 3. On y observe une mémoire DRAM principale reliée à la mémoire cache L1 de 64 KB. La cache L1 est reliée à un nombre paramétrable de *Lanes*. Chaque *Lane* comporte sa banque de 8 registres et son unité de *multiply-accumulate*.

Afin de trouver les meilleurs mappings, la

commande *timeloop-mapper* a été utilisée. L'optimisation du mapping se fait en utilisant comme seule métrique l'énergie. L'ensemble des résultats ont étés obtenues en utilisant la technologie 45 nm fournie par Accelergy. Aussi, un problème sur le *victory-condition* du mapping a été trouvé en utilisant *timeloop-mapper*. Ainsi, le code source de Timeloop a été modifié afin d'obtenir des résultats. Le problème est relié à un *flag* qui n'était pas correctement configuré. Ainsi, le compteur du *victory-condition* ne s'incrémentait pas correctement.

Afin d'étudier la performance de l'architecture en fonction du nombre de *Lanes*, les problèmes utilisés sont les trois premières couches convolutionnelles de VGG16 et les trois premières couches convolutionnelles d'AlexNet. Ces deux groupes de problèmes ont été choisis pour observer la variation de la taille du problème, du nombre de canaux, le nombre de filtres et la taille des *kernels*. Avec VGG16, chaque couche convolutionnelle possède la même taille de *kernel*, mais varie en canaux, filtres et taille. Pour AlexNet, nous avons trois différentes tailles de *kernels*. La figure 4 illustre les dimensions des problèmes étudiés.

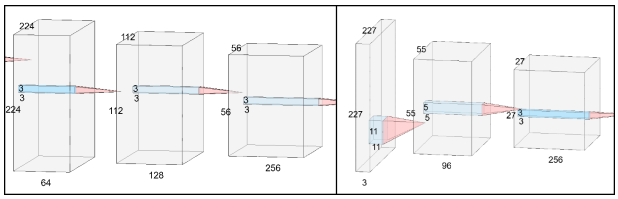

FIGURE 4 – Problèmes étudiés : VGG16 (gauche) et AlexNet (droite)

Afin de mesurer la performance énergétique, la métrique utilisée est le GFLOP/J. La formule utilisée pour calculer cette métrique est la suivante :

$$\text{GFLOP/J} = \frac{\%utilization \times \#cycles \times \#lanes}{E_{tot}}$$

L'ensemble des fichiers sources utilisés afin d'obtenir les résultats sont disponibles sur le repo GitHub suivant : https://github.com/YoanFournier/projet_ELE6307.

3 Résultats

Cette section présente les résultats d'aire, de cycles, d'énergie et de performance en fonction du nombre de *Lanes* pour les 3 premières couches de AlexNet et VGG16.

Tout d'abord, la figure 11 montre que l'aire du circuit augment de façon linéaire en fonction du nombre de *Lanes*. C'est un résultat attendu puisque chaque *Lane* prend, à priori, la même surface. Il reste important de comprendre que ce résultat est approximatif, en réalité, la complexité du P&R risque d'augmenter plus rapidement que de façon linéaire en fonction du nombre de *Lanes*.

Les figures 5 et 6 montrent la corrélation entre le nombre de cycles pour effectuer le calcul des couches convolutive en fonction du nombre de *Lanes*. Bien que l'on pourrait s'attendre à une réduction linéaire des cycles, on remarque que ce n'est pas tout à fait le cas. En effet, l'utilisation des *Lanes* n'est pas de 100% pour tous les problèmes comme le démontre la rangée *Utilization* des tableaux 2 et 3. Le mapper de Timeloop n'arrive donc pas toujours à trouver un mapping utilisant plus de ressources qui optimise la consommation d'énergie.

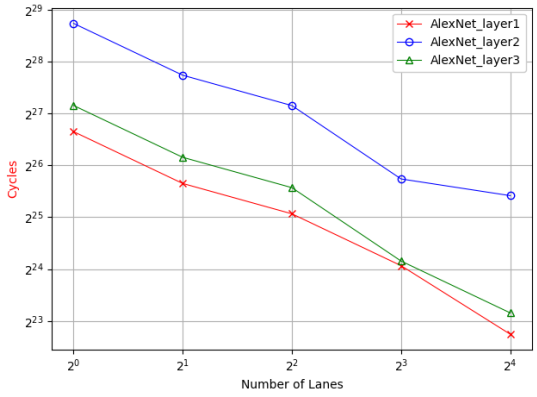

FIGURE 5 – Cycles de ARA en fonction du nombre de *Lanes* pour le problème AlexNet

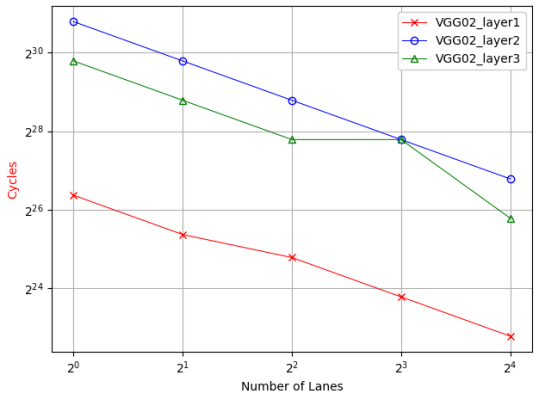

FIGURE 6 – Cycles de ARA en fonction du nombre de *Lanes* pour le problème VGG16

Les figures 7 et 8 présentent l'énergie consommée pour les différents calculs en fonction du nombre de *Lanes*. On remarque une tendance à la baisse dans la consommation d'énergie plus le nombre de *Lanes* est élevé. En effet, plus il y a de *Lanes*, plus il y a de banques de registres disponibles. Ainsi, la quantité de mémoire locale aux unités de calcul est plus élevée et il est moins nécessaire d'avoir accès aux hiérarchies de mémoires plus élevées comme la cache ou la DRAM. Ceci à pour effet de réduire la consommation d'énergie. Il reste que ces gains énergétiques sont assez faibles et dans l'ensemble le meilleur gain d'énergie observé est de 1.52% pour la couche 2 de VGG16 entre une *Lane* et 16 *Lanes*.

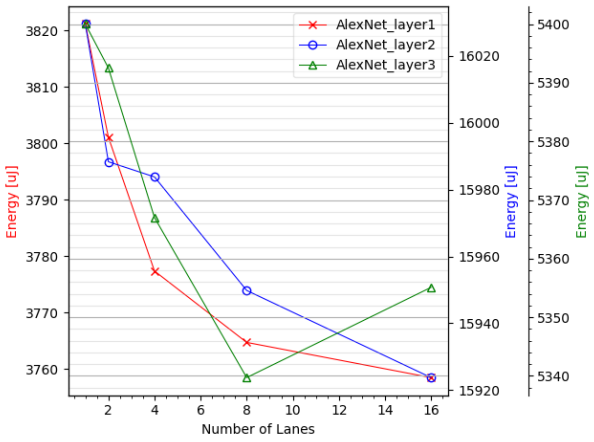

FIGURE 7 – Énergie de ARA en fonction du nombre de *Lanes* pour le problème AlexNet

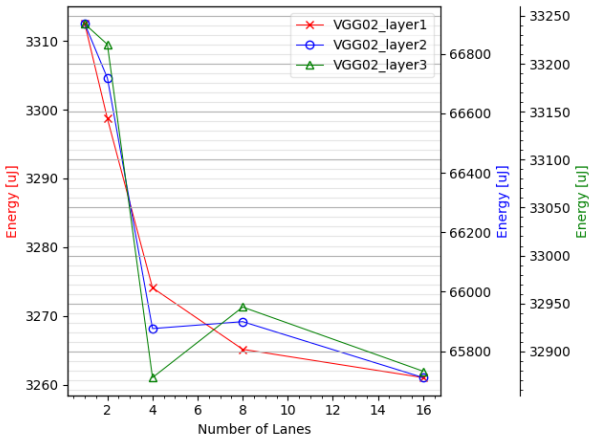

FIGURE 8 – Énergie de ARA en fonction du nombre de *Lanes* pour le problème VGG16

Les figures 9 et 10 mettent en avant l'efficacité énergétique en GFLOP/J en fonction du nombre de *Lanes*. Le même constat que pour la consommation énergétique peut-être observée. Il y a une tendance à la hausse de l'efficacité,

mais les gains sont minimes et restent plutôt constants. On observe au maximum, un gain de 1.92% entre une *Lane* et 16 *Lanes* pour la couche 1 de AlexNet.

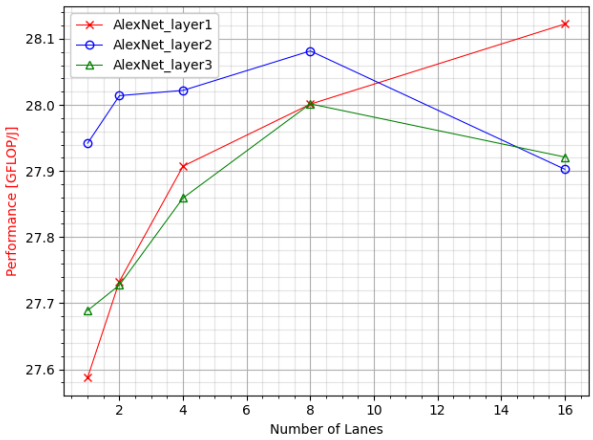

FIGURE 9 – Performance de ARA en fonction du nombre de *Lanes* pour le problème AlexNet

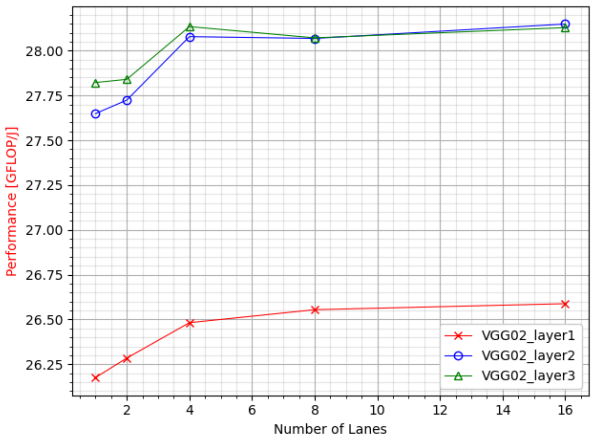

FIGURE 10 – Performance de ARA en fonction du nombre de *Lanes* pour le problème VGG16

À la lumière de ces résultats, un aspect intéressant peut être soulevé. On remarque qu'en général, le mapper de Timeloop semble préférer effectuer le mapping spatial sur la composante C du problème, soit le nombre de canaux d'entrée. La rangée *Utilization* du tableau 2 le démontre bien. On remarque une utilisation de 75 %, 75 % et 94 % pour 4, 8 et 16 *Lanes* respectivement. Ceci correspond à une utilisation de 3, 6 et 15 *Lanes* qui sont tous des multiples de C=3. L'avantage de faire la distribution spatiale des données selon C est efficace énergétiquement puisque l'ensemble des canaux sont indépendants. Chaque *Lane* peut donc effectuer ses calculs de façon indépendante des autres ce qui permet de limiter les transferts mémoire entre les *Lanes* et à comme effet de minimiser la consommation énergétique. Comme la plupart des couches internes d'un DNN possèdent un nombre pair de canaux d'entrée, il serait sans doute intéressant d'utiliser un mapping spatial sur C dans la majorité des cas. Cependant, on remarque dans les tableaux 2 et 3 que l'utilisation n'est pas de 100 % pour les problèmes ayant un C pair.

Problem	**AlexNet_layer2**	
Parameters	M=256 K=5x5 C=96 I=27x27	
n	16	
Mapper	Timeloop	Custom
Utilization	0.62	1
Cycles [Millions]	44.8	28.0
Energy [mJ]	15.92	15.90
Perf [GFLOP/J]	27.90	28.18

TABLEAU 1 – Comparaison entre un mapping Timeloop et un mapping custom pour AlexNet L2

Afin de tester l'hypothèse qu'un mapping spatial sur C est généralement plus intéressant, un mapping custom a été conçu afin de le comparer au mapping effectué par Timeloop. Le problème utilisé est la couche 2 d'AlexNet sur 16 *Lanes*. Les deux mappings sont présentés au fi-

gures 12 et 13. La couche 2 d'AlexNet comporte 96 canaux, ce qui est divisible par le nombre de *Lanes*. Cependant, le mapping de Timeloop n'utilise que 62 % des *Lanes* comme le montre le tableau 1. En utilisant un mapping custom faisant le mapping spatial sur C, on obtient une utilisation de 100 % ainsi qu'un gain énergétique et un gain en efficacité.

En général, les résultats démontrent clairement l'avantage que peut apporter un processeur vectoriel. En effet, bien que la consommation et l'efficacité énergétique restent similaires pour les différents nombres de *Lanes*, il faut comprendre qu'augmenter le nombre de *Lanes* réduit le nombre de cycles nécessaires afin d'effectuer un calcul. En effet, si l'on utilise 100 % des ressources, l'utilisation de 16 *Lanes* apporte un gain en performance jusqu'à un facteur 16 par rapport à un processeur scalaire et ceci pour une efficacité énergétique identique, voire meilleure.

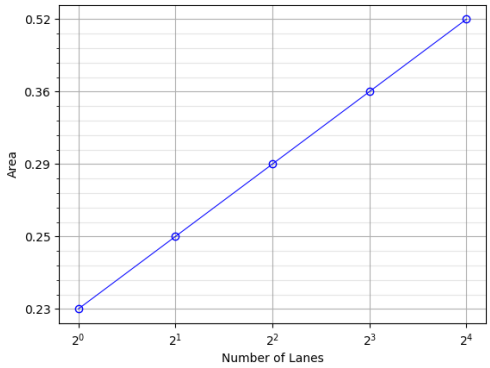

FIGURE 11 – Aire de ARA en fonction du nombre de *Lanes*

4 Conclusion

Nous visions à comparer l'efficacité d'un processeur vectoriel contre un processeur scalaire dans le cas de calculs utilisés dans les DNNs afin de démontrer l'avantage d'utiliser des processeurs vectoriels. Nous avons étudié la consommation totale, le nombre de cycles, l'aire et l'efficacité énergétique en variant le nombre de *Lanes* dans notre architecture pour six problèmes différents. Nous avons pu démontrer qu'augmenter le nombre de *Lanes* permet d'atteindre de meilleures performances en temps sans compromettre l'efficacité énergétique du processeur, voire permettre une efficacité légèrement meilleure. En effet, augmenter le nombre de *Lanes* permet la parallélisation des calculs, permettant des gains linéaires dans le cas de données indépendantes. Cet effet est indépendant du problème, pour autant qu'un mapping adéquat est utilisé. La génération de ces mappings est cependant complexe et nécessiterait une étude plus approfondie et une meilleure connaissance du problème et de l'architecture. Nous pouvons donc affirmer avoir atteint nos objectifs initiaux.

Dans le futur, il serait important de modéliser le séquenceur et le SLDU d'ARA. Ces composantes exécutent un travail complexe qui ne peut pas être ignoré pour avoir un modèle complet du flot de données dans un processeur vectoriel. Modéliser ces unités pourrait révéler un coût caché venant avec l'augmentation du nombre de *Lanes*. Pour améliorer la recherche de mappings au travers de Timeloop, il serait pertinent de modifier le critère d'optimisation du mapper pour l'efficacité (GFLOP/J) ou de contraindre davantage le mapper pour encourager un mapping spacial maximal sur les canaux. Ces modifications devraient être sommaires à réaliser puisque Timeloop est open source.

Références

[1] B. Bougenot. Ara : Update pulp's vector processor. Master's thesis, Swiss Federal Institute of Technology in Zürich (ETH), 2020.

[2] M. Cavalcante, F. Schuiki, F. Zaruba, M. Schaffner, and L. Benini. Ara : A 1 ghz+ scalable and energy-efficient risc-v vector processor with multi-precision floating point support in 22 nm fd-soi, 2019.

Annexe

```
OFFCHIP_MEM [ Weights:614400 (614400) Inputs:92256 (92256) Outputs:186624 (186624) ]
-------------------------------------------------------------------------------------
| for P in [0:3)
|   for M in [0:64)

L1_CACHE [ Inputs:38688 (38688) Outputs:972 (972) ]
---------------------------------------------------
|     for C in [0:24)
|       for P in [0:9)
|         for Q in [0:27)
|           for R in [0:5) (Spatial-X)
|             for M in [0:2) (Spatial-X)

BANK [ Weights:40 (40) Inputs:20 (20) Outputs:2 (2) ]
-----------------------------------------------------
|               for S in [0:5)
|                 for M in [0:2)
|                   for C in [0:4)
```

FIGURE 12 – Mapping Timeloop pour AlexNet L2

```
OFFCHIP_MEM [ Weights:614400 (614400) Inputs:92256 (92256) Outputs:186624 (186624) ]
-------------------------------------------------------------------------------------
| for M in [0:8)
|   for C in [0:3)

L1_CACHE [ Inputs:30752 (30752) Outputs:23328 (23328) ]
-------------------------------------------------------
|     for M in [0:8)
|       for R in [0:5)
|         for C in [0:2)
|           for P in [0:9)
|             for Q in [0:27)
|               for C in [0:16) (Spatial-X)

BANK [ Weights:20 (20) Inputs:15 (15) Outputs:12 (12) ]
-------------------------------------------------------
|                 for P in [0:3)
|                   for S in [0:5)
|                     for M in [0:4)
```

FIGURE 13 – Mapping custom pour AlexNet L2

Problem	**AlexNet_layer1**					**AlexNet_layer2**					**AlexNet_layer3**				
Parameters	M=96 K=11x11 C=3 I=55x55					M=256 K=5x5 C=96 I=27x27					M=384 K=3x3 C=256 I=13x13				
n	1	2	4	8	16	1	2	4	8	16	1	2	4	8	16
Utilization	1	1	0.75	0.75	0.94	1	1	0.75	1	0.62	1	1	0.75	1	1
Cycles [Millions]	105.4	52.7	35.1	17.6	7.3	447.9	223.9	149.3	56.0	44.8	149.5	74.8	49.8	18.7	9.3
Energy [mJ]	3.82	3.80	3.77	3.77	3.76	16.03	15.99	15.98	15.95	15.92	5.40	5.39	5.36	5.34	5.35
Perf [GFLOP/J]	27.59	27.73	27.91	28.00	28.12	27.94	28.01	28.02	28.08	27.90	27.69	27.73	27.86	28.00	27.92

TABLEAU 2 – Résultats pour le problème AlexNet

[Problem	**VGG02_layer1**					**VGG02_layer2**					**VGG02_layer3**				
Parameters	M=64 K=3x3 C=3 I=224x224					M=64 K=3x3 C=64 I=224x224					M=128 K=3x3 C=64 I=112x112				
n	1	2	4	8	16	1	2	4	8	16	1	2	4	8	16
Utilization	1	1	0.75	0.75	0.75	1	1	1	1	1	1	1	1	0.5	1
Cycles [Millions]	86.7	43.4	28.9	14.5	7.2	1850	924.8	462.4	231.2	115.6	925	462	231	231	57.8
Energy [mJ]	3.31	3.30	3.27	3.27	3.26	66.9	66.7	65.9	65.9	65.7	33.2	33.2	32.9	32.9	32.9
Perf [GFLOP/J]	26.18	26.28	26.48	26.55	26.59	27.65	27.72	28.08	28.07	28.15	27.82	27.84	28.13	28.07	28.13

TABLEAU 3 – Résultats pour le problème VGG16