Introduction d'aléas dans le processus de projection d'applications sur CGRA

Satyajit Das, Kevin Martin, Thomas Peyret, Philippe Coussy

Compas'2016 : Parallélisme/ Architecture / Système
Lorient, France, du 5 au 8 juillet 2016

Introduction d'aléas dans le processus de projection d'applications sur CGRA

Satyajit Das[1], Kevin J. M. Martin[1], Thomas Peyret[2], Philippe Coussy[1]

[1]Univ. Bretagne-Sud, UMR CNRS 6285, Lab-STICC, F-56100 Lorient, France

[2]CEA, LIST, F-91191 Gif-sur-Yvette, France

Résumé
Les architectures reconfigurables à gros grains offrent un compromis flexibilité-performance intéressant à travers les nombreuses unités de calculs élémentaires qu'elles proposent. Cependant, projeter automatiquement une application sur une architecture reconfigurable à gros grain est un processus complexe qui nécessite d'explorer un vaste espace de solutions. Cet article propose d'étudier l'apport d'aléas dans le processus de projection. L'introduction d'aléas est effectué en particulier dans les étapes d'ordonnancement et d'assignation. Différentes stratégies permettant de garantir un nombre minimum et maximum de solutions sont présentées. Les résultats montrent que notre méthode, couplée à une approche de transformation du graphe d'application, explore mieux l'espace de solutions et permet de trouver la latence la plus courte.

Mots-clés : CGRA, projection, aléatoire, assignation, ordonnancement

1. Introduction

Depuis deux décennies maintenant, les architectures reconfigurables à gros grains sont étudiées pour leur compromis intéressant entre flexibilité et efficacité énergétique. La littérature regorge d'architectures de type CGRA [14, 13], notamment pour la famille des applications multimedia. Ces architectures se distinguent par la granularité des éléments de calculs, leur homogénéité, les opérations supportées ou le réseau d'interconnexion. L'élément de calcul (*Processing Element*, PE), appelé tuile, est habituellement constitué d'une unité fonctionnelle (*Functional Unit*, FU), d'une file de registres (*Register File*, RF) et d'un registre de sortie comme illustré par la figure 1. Le Samsung Reconfigurable Processor (SRP) [6], intégré dans le SoC Exynos [3], est certainement l'exemple commercial le plus abouti d'un CGRA. Les succès sont finalement limités à cause du manque d'outils de compilation efficaces pour projeter automatiquement les applications sur ces architectures. La forte adhérence entre les techniques de projection d'applications et les architectures CGRA, elles-mêmes conçues spécifiquement pour une famille d'applications, exacerbe la difficulté.
Compiler une application sur un CGRA consiste à trouver une solution valide d'ordonnancement et d'assignation des opérations et des variables sur les ressources du CGRA tout en respectant les dépendances de données et de contrôle et les contraintes architecturales du CGRA. Le résultat de ce processus est nommé projection. La complexité croissante des applications pousse à automatiser ce processus de projection. Malheureusement, l'ordonnancement et l'assignation sont inter-dépendants et sont des problèmes NP-complets. Si trouver une projection valide reste un problème complexe, trouver une projection valide qui minimise certains critères (comme la latence ou la surface par exemple) est un véritable défi. Les compilateurs s'appuient traditionnellement sur une représentation de l'application sous forme de graphes de type CDFG (Control Data-Flow Graph), qui représente à la fois les dépendances de données et de contrôle.
Les CGRAs étant initialement conçus pour accélérer des cœurs de boucles, les méthodes de projection de la littérature s'intéressent principalement à minimiser l'intervalle d'initiation afin de maximiser le débit [7, 9, 8, 2, 15]. Dans nos travaux, nous cherchons la projection dont la latence est la plus courte possible. De manière générale, les approches de projection peuvent être classifiées en deux grandes

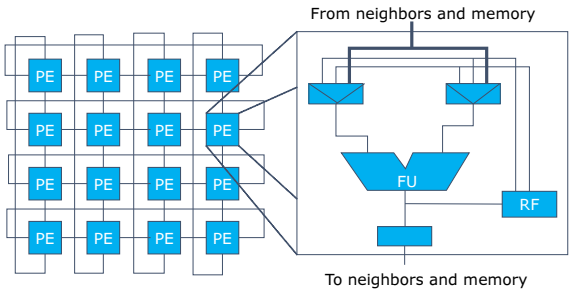


FIGURE 1 – Illustration d'un CGRA 4×4 avec une unité fonctionnelle (FU) et une file de registre (RF) dans chaque tuile (PE) en topologie mesh 2D torique

catégories. Elles diffèrent selon la façon dont les problèmes d'assignation et d'ordonnancement sont résolus : séquentiellement ou conjointement. La première catégorie regroupe les travaux présentés dans [7, 9, 4, 5]. La seconde catégorie inclut [1, 8, 2, 10, 11, 12]. Les deux problèmes étant inter-dépendants, la résolution conjointe est la meilleure approche mais elle est beaucoup plus difficile à maîtriser.

Ce travail s'appuie sur nos précédents travaux qui proposent une résolution conjointe [11]. Dans ce papier, nous proposons d'étudier l'apport d'aléas dans le processus de projection, et notamment dans les étapes d'ordonnancement et d'assignation. L'utilisation d'aléas nous permet de guider la méthode pour conserver un nombre restreint de solutions partielles. Une solution partielle est un résultat d'ordonnancement et d'assignation valide pour un cycle donné. Une solution complète se construit de manière incrémentale à partir d'une solution partielle. Le nombre de solutions partielles est gigantesque et seules quelques unes d'entre elles mènent à une projection valide. Il s'agit donc de conserver suffisamment de solutions partielles pour aboutir à une solution complète tout en maîtrisant leur nombre pour passer à l'échelle. La méthode est éprouvée sur des CGRAs possédant différents paramètres architecturaux : nombre de tuiles, taille de la file de registre, type d'interconnexion. Les résultats montrent que la combinaison de l'ensemble permet d'atteindre les mêmes performances avec un CGRA plus simple en terme architecture et donc moins coûteux en surface qu'un CGRA plus complexe.

La méthode est présentée dans la section 2. Les résultats et leurs interprétations sont fournis dans la section 3. La section 4 conclut cet article.

2. Une approche basée sur l'aléa

L'approche proposée s'appuie fortement sur les travaux présentés dans [11], où les problèmes d'assignation et d'ordonnancement sont résolus conjointement, grâce à un parcours topologique inverse et des transformations de graphe. Après chaque cycle d'ordonnancement, une étape d'assignation exhaustive permet de connaître toutes les solutions partielles au cycle courant. Le nombre de solutions partielles devient alors très vite important. Dans [11], il est proposé de supprimer les solutions redondantes, c'est-à-dire toutes les projections similaires. Cette technique d'élagage souffre d'un problème de passage à l'échelle. Nous proposons dans cet article d'introduire une part d'aléas dans la conservation des solutions partielles (étape d'élagage) et dans la construction des solutions partielles (étape d'ordonnancement).

2.1. Introduction de l'aléa dans l'étape d'assignation

Une approche naïve serait de conserver un simple pourcentage de projections. Mais ce n'est pas judicieux, surtout en début de résolution du processus de projection. En effet, si seulement deux projections partielles existent, en supprimer une des deux peut simplement aboutir à un échec. Le nombre de projections à conserver dépend donc du nombre de projections partielles au cycle courant. Moins il y a de solutions partielles au cycle courant, plus il faut en conserver. Pour déterminer si une projection partielle doit être conservée, un nombre aléatoire compris entre 0 et 1 est tiré. Si ce nombre est inférieur à un certain seuil, la projection est conservée, sinon elle est rejetée. Ce seuil n'est donc pas fixe, il doit s'adapter en fonction du nombre de projections (nbM). Pour cela, ce nombre de projections est normalisé suivant

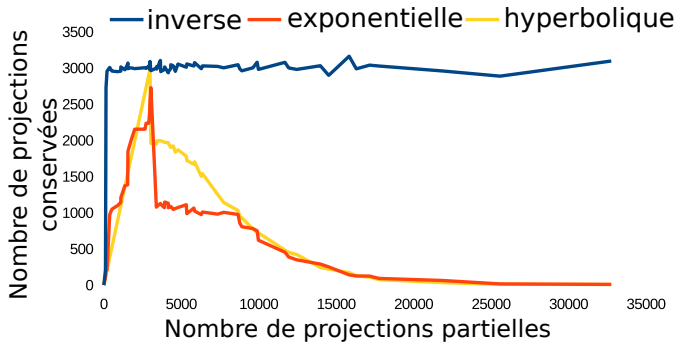


FIGURE 2 – Étude de plusieurs fonctions de seuil, la fonction inverse est sélectionnée pour sa stabilité

un paramètre λ, défini par l'utilisateur, et une fonction mathématique.

2.1.1. Trouver la bonne fonction mathématique

La fonction mathématique à considérer est une fonction décroissante : plus le nombre de projections est petit, plus on en garde, plus ce nombre est grand, moins on en conserve. Nous avons mené des expérimentations sur la fonction exponentielle inverse (typiquement utilisée dans le recuit simulé [2, 8]), la fonction hyperbolique, et la fonction inverse. La figure 2 montre le nombre moyen de projections partielles conservées pour dix exécutions pour une valeur de lambda de 3000. Les résultats montrent que la fonction inverse permet de conserver en moyenne λ projections, alors que les fonctions exponentielle et hyperbolique ne permettent pas de conserver un nombre constant de projections. La fonction inverse est alors sélectionnée pour définir la fonction de seuil (Equ. 1). Nous avons observé la même tendance pour plusieurs valeurs du paramètre λ.

$$\text{Seuil}(\text{nbM}, \lambda) = \begin{cases} (\lambda/\text{nbM}) & si\ \text{nbM} > \lambda \\ 1 & si\ \text{nbM} \leq \lambda \end{cases} \tag{1}$$

Le paramètre λ est à ajuster avec soin puisqu'il impacte directement les chances d'obtenir finalement une projection valide et le temps de compilation. Plus λ est grand, plus le nombre de projections conservées est grand, conduisant à de meilleures chances de succès mais aussi à des temps de compilation plus grands et une empreinte mémoire élevée. La valeur de 3000 pour λ s'avère présenter le bon compromis en obtenant des projections valides à chaque fois dans des temps de compilation raisonnables (quelques dizaines de secondes).

2.1.2. Encadrement du nombre de projections par des bornes hautes ou basses

Le contrôle du nombre de projections partielles à chaque cycle (nbCM) est donc primordial, notamment un nombre minimum. Nous proposons d'introduire une borne basse et une borne haute pour encadrer le nombre de projections. Nous présentons deux stratégies d'utilisation de ces bornes :

1. Borne basse seulement (BBs). Pour chaque projection partielle, un nombre aléatoire compris entre 0 et 1 est tiré et comparé à la fonction de seuil. Ce processus de sélection continue tant que le nombre minimum de projection, défini par l'Equ. 2 n'est pas atteint.

$$\min \text{nbCM} = \lceil \text{nbM}/\lambda \rceil \tag{2}$$

2. Borne basse et borne haute (BB+BH). Pour chaque projection partielle, un nombre aléatoire compris entre 0 et 1 est tiré et comparé à la fonction de seuil. Ce processus s'arrête (même avant d'avoir parcouru toutes les solutions) si le nombre de projections conservées atteint le nombre maximum autorisé.

$$\max \text{nbCM} = \lceil \text{nbM}/3 \rceil \tag{3}$$

Benchmark	nombre de nœuds opérations	ASAP	Parallélisme
DCT-2D	711	81	32
matrix product	504	98	32
FFT	1348	37	64
Trapezoidal filter	332	59	32
EMA filter	412	99	38
MWD filter	440	112	32
Unsharp Mask	91	27	16
Elliptic Filter	130	31	16
DC Filter	507	96	32

TABLE 1 – Codes applicatifs considérés et leurs caractéristiques

$$\min \mathrm{nbCM} = \begin{cases} \lfloor \mathrm{nbM}/\lambda \rfloor & \text{if } \mathrm{nbM} > \lambda \\ \lceil \mathrm{nbM}/3 \rceil & \text{if } \mathrm{nbM} \leq \lambda \end{cases} \quad (4)$$

La stratégie consistant à vérifier que le nombre de projections conservées est inférieur à une borne haute seulement n'est pas retenue. En effet, il faut principalement garantir un nombre minimum de projections partielles afin d'augmenter les chances d'obtenir au moins une projection valide.

2.2. Introduction de l'aléa dans l'étape d'ordonnancement

L'ordonnancement utilisé dans [11] est un ordonnancement par listes avec un parcours topologique inverse. Cet heuristique propose d'ordonnancer les nœuds selon une fonction de priorité. Deux critères de tri sont utilisés dans notre approche : 1) la mobilité des nœuds, 2) le nombre d'arcs sortants pour les nœuds ayant la même mobilité. Malgré ces deux critères de tri, il se peut que plusieurs nœuds aient la même priorité (typiquement ceux ayant même mobilité et un seul arc sortant).
Selon la mise en œuvre de cette méthode d'ordonnancement et particulièrement les structures de données utilisées, il se peut que cet ordonnancement soit déterministe [1], alors qu'aucun critère ne permet a priori de différencier plusieurs nœuds ordonnançables ayant même priorité. Nous proposons d'introduire de l'aléa à ce stade. La mise en œuvre est simple puisqu'il s'agit de prendre au hasard un nœud dans la liste (des nœuds ayant strictement même priorité bien sûr), et non celui placé en tête de liste lors de la construction des structures de données.

3. Résultats

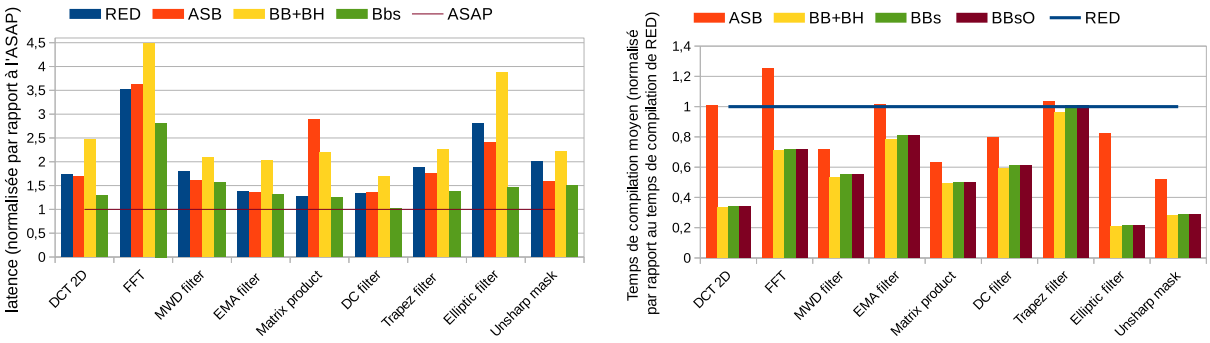
3.1. Environnement expérimental

L'ajout d'aléas dans la méthode de projection a été mise en œuvre dans notre outil. GCC 4.8 est utilisé pour générer les CDFG à partir de codes applicatifs en langage C. Neufs applications issues du domaine du traitement du signal, présentées dans le tableau 1, ont été utilisées pour nos expérimentations. La station de travail utilisée contient quatre CPU Intel Xeon @ 3.5 GHz et 8 Go de mémoire RAM.
Nous avons étudié plusieurs cas de figures pour apprécier l'apport d'aléas dans la méthode de projection. Toutes les méthodes s'appuient sur le parcours en sens inverse des nœuds, l'ordonnancement et l'assignation simultanées, les transformations de graphes. Elles diffèrent dans leur manière d'élaguer l'espace de solutions : (i) *RED (suppression des **Redondants**)* [10] supprime seulement les projections redondantes. (ii) *ASB* (aléatoire sans bornes) utilise l'aléa sans assurer un nombre minimum ou maximum de projections conservées. (iii) *BB+BH* (Borne Basse et Borne Haute) utilise l'aléa mais garantit un nombre minimum et maximum de projections conservées. (iv) *BBs* (Borne Basse Seulement) utilise l'aléa en garantissant un nombre minimum de projections conservées (pas de limite sur le nombre maximum).

3.2. Résultats

Nous avons commencé par étudier l'apport d'aléas dans la méthode d'assignation. Puisque la méthode est aléatoire, nous avons exécuté dix fois chaque stratégie et présentons les meilleurs résultats de chacune. Ce premier jeu d'expérimentations nous permet de fixer la stratégie pour l'assignation. Nous présentons ensuite l'apport d'aléas dans la méthode d'ordonnancement.

1. ce qui est le cas pour notre outil



(a) Latence (b) Temps de compilation

FIGURE 3 – Comparaison des approches avec l'aléa dans l'assignation

3.2.1. Ajout de l'aléa dans l'assignation

Pour comparer les différentes méthodes, nous avons d'abord considéré un CGRA de 3x3 homogène possédant une file de registre de 24 (cette taille pour la file de registre a déjà été montrée suffisante pour ne pas faire apparaître des problèmes d'assignation des données). Les figures 3a et 3b montrent respectivement les résultats de latence des codes applicatifs considérés et les temps de compilation. La latence obtenue est normalisé par rapport à la plus courte latence possible (ASAP, *As Soon As Possible*). Les temps de compilation sont normalisés par rapport à la méthode initiale RED, qui supprime les projections redondantes. La figure 3a montre les mêmes tendances pour les différentes stratégies. C'est la stratégie BBs qui mène aux meilleurs résultats alors que la stratégie BB+BH fournit les pires. Concernant la FFT, nous avons mené d'autres expérimentations avec un CGRA d'une taille 6x6 pour observer les mêmes tendances et obtenir une projection dont la latence est égale à l'ASAP.
Les temps de compilation en figure 3b montrent que les deux stratégies BBs et BB+BH sont les plus rapides. La stratégie BBs donne de meilleurs résultats que BB+BH. Elle est donc conservée pour la suite des expérimentations.

3.2.2. Ajout de l'aléa dans l'ordonnancement

La méthode BBs avec l'ajout d'aléas dans l'ordonnancement est appelée BBsO. L'ajout d'aléas dans l'ordonnancement nous permet de mieux explorer l'espace de solutions. Ceci est illustré par la figure 4. Cette figure montre les résultats des dix exécutions des trois méthodes RED, BBs, et BBsO pour le code IDCT pour différents paramètres architecturaux du CGRA. Chaque point de la figure correspond à une exécution de chaque méthode. L'axe des abscisses représente la latence normalisée par rapport à l'ASAP, l'axe des ordonnées représente le nombre de nœuds transformés par rapport au nombre de nœuds initial. Il apparaît clairement sur la figure que l'aléa dans l'ordonnancement permet de couvrir un plus grand espace de solutions. Dans la majorité des cas, les résultats de latence sont meilleurs avec aléa que sans aléa dans l'ordonnancement. Les mêmes tendances ont été observées pour les autres codes applicatifs. Cet ajout d'aléas est sans impact sur les temps de projection (voir figure 3b).

3.2.3. Influence des paramètres architecturaux du CGRA

Afin de tester la robustesse de notre approche, nous avons conduit des expérimentations en faisant varier les caractéristiques architecturales du CGRA. Les figures 6a et 6b montrent les résultats en termes de latence pour un CGRA de taille 4x4, avec différentes tailles de RF (8, 16, ou 24), et différentes topologies (mesh-torus, mesh-X-Torus, ou entièrement connecté) illustrées par la figure 5. La figure 6a montre les résultats obtenus pour une bande passante infinie, c'est-à-dire que le nombre d'accès à la mémoire n'est pas limitée. La figure 6b montre les résultats pour un nombre limité d'unités de load/Store : 4 (LS 4) ou 8 (LS 8). Les résultats montrent que finalement, un CGRA complexe, avec un vaste réseau d'interconnexion et un grand nombre de ressources n'apporte pas de performance significative comparé à un CGRA plus modeste, possédant un simple réseau en mesh-2D-torus, avec une file de registres de taille 8. Les expérimentations menées sur d'autres tailles de CGRA mais non illustrées ici par manque de place

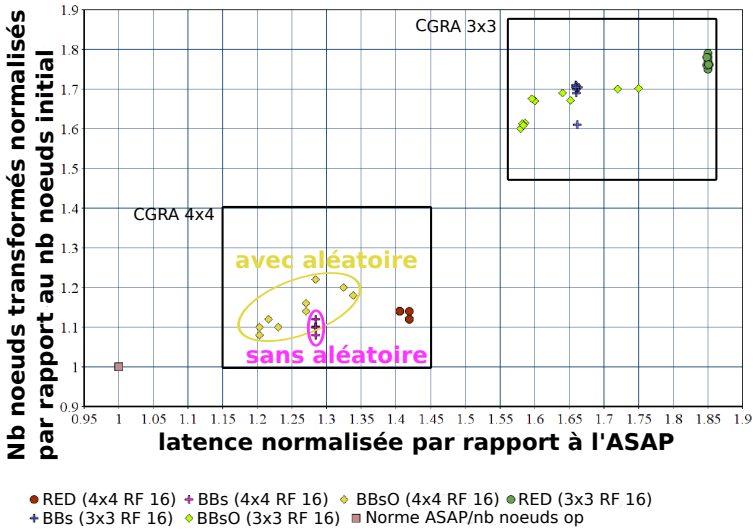


FIGURE 4 – Couverture de l'espace de solution avec l'aléa dans l'ordonnancement

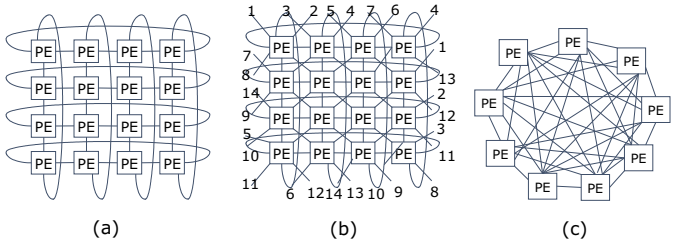


FIGURE 5 – Topologies : (a) mesh torus, (b) mesh-x-torus, (c) entièrement connecté

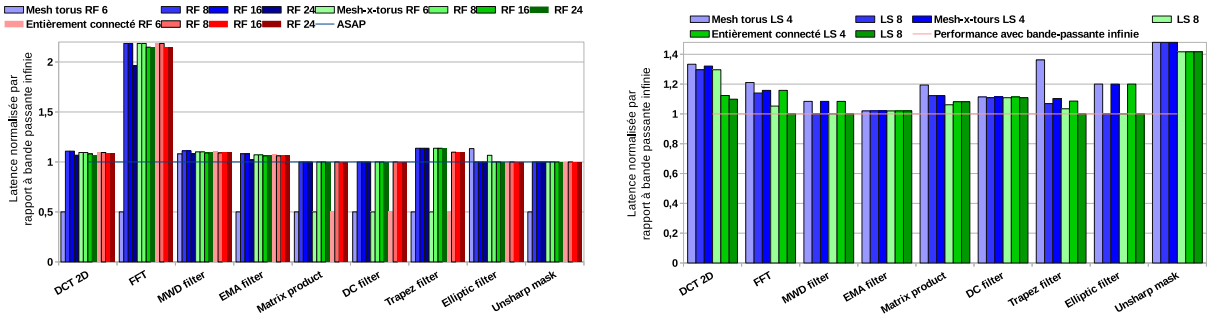
permettent d'observer la même tendance.
Ces bons résultats sur des CGRAs simples sont à mettre au crédit de l'ensemble de l'approche de projection que nous proposons, dont l'ajout d'aléas et la transformation du graphe d'application en fonction de l'architecture. Pour observer ces transformations, il faudrait illustrer le nombre de nœuds transformés (comme dans la figure 4). Par manque de place, ces résultats ne sont pas fournis, mais sans surprise, le nombre de transformations est plus important pour des CGRAs simples que des CGRAs complexes.

4. Conclusion

Dans cet article, nous présentons l'apport d'aléas dans une méthode de projection d'applications sur CGRA. L'ajout d'aléas est d'abord introduit dans l'étape d'assignation, puis dans l'étape d'ordonnancement. Les résultats montrent que l'introduction d'aléas permet de mieux explorer l'espace des solutions. La méthode permet d'atteindre la latence minimum pour les applications avec des temps de compilation raisonnables (quelques dizaines de secondes). La combinaison de tous les aspects de l'approche : parcours topologique inverse, transformation du graphe à la volée, ordonnancement aléatoire et élagage aléatoire avec borne basse permet de trouver des projections ayant les mêmes performances sur des CGRAs simples que des CGRAs complexes en termes d'architecture.

Bibliographie

1. Brenner (J.), van der Veen (J.), Fekete (S.), Oliveira Filho (J.) et Rosenstiel (W.). – Optimal Simultaneous Scheduling, Binding and Routing for Processor-Like Reconfigurable Architectures. – In *FPL*, pp. 1–6, août 2006.



(a) Bande passante infinie

(b) Bande passante limitée

FIGURE 6 – Comparaison des latences pour un CGRA 4x4

2. De Sutter (B.), Coene (P.), Vander Aa (T.) et Mei (B.). – Placement-and-routing-based register allocation for coarse-grained reconfigurable arrays. *SIGPLAN Not.*, vol. 43, n7, juin 2008, pp. 151–160.
3. Frumusanu (A.). – The Samsung Exynos 7420 Deep Dive - Inside A Modern 14nm SoC.
4. Hamzeh (M.), Shrivastava (A.) et Vrudhula (S.). – EPIMap : using epimorphism to map applications on CGRAs. – In *DAC*, pp. 1284–1291. ACM, 2012.
5. Hamzeh (M.), Shrivastava (A.) et Vrudhula (S.). – REGIMap : register-aware application mapping on coarse-grained reconfigurable architectures (CGRAs). – In *DAC*, pp. 18 :1–18 :10. ACM, 2013.
6. Kim (C.), Chung (M.), Cho (Y.), Konijnenburg (M.), Ryu (S.) et Kim (J.). – ULP-SRP : Ultra low power Samsung Reconfigurable Processor for biomedical applications. – In *FPT*, pp. 329–334, décembre 2012.
7. Lee (G.), Choi (K.) et Dutt (N.). – Mapping Multi-Domain Applications Onto Coarse-Grained Reconfigurable Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, n5, mai 2011, pp. 637–650.
8. Mei (B.), Vernalde (S.), Verkest (D.), De Man (H.) et Lauwereins (R.). – DRESC : a retargetable compiler for coarse-grained reconfigurable architectures. – In *FPT*, pp. 166–173, décembre 2002.
9. Park (H.), Fan (K.), Mahlke (S. A.), Oh (T.), Kim (H.) et Kim (H.-s.). – Edge-centric Modulo Scheduling for Coarse-grained Reconfigurable Architectures. – In *PACT*. ACM, 2008.
10. Peyret (T.), Corre (G.), Thevenin (M.), Martin (K.) et Coussy (P.). – Efficient application mapping on CGRAs based on backward simultaneous scheduling/binding and dynamic graph transformations. – In *ASAP*, pp. 169–172, juin 2014.
11. Peyret (T.), Corre (G.), Thevenin (M.), Martin (K.) et Coussy (P.). – Ordonnancement, assignation et transformations dynamiques de graphe simultanés pour projeter efficacement des applications sur CGRAs. – In Pascal Felber, Laurent Philippe (E. R. A. T.) (édité par), *ComPAS 2014 : conférence en parallélisme, architecture et systèmes*, Neuchatel, Switzerland, avril 2014.
12. Raffin (E.), Wolinski (C.), Charot (F.), Kuchcinski (K.), Guyetant (S.), Chevobbe (S.) et Casseau (E.). – Scheduling, binding and routing system for a run-time reconfigurable operator based multimedia architecture. – In *DASIP*, pp. 168–175, 2010.
13. Sutter (B. D.), Raghavan (P.) et Lambrechts (A.). – Coarse-Grained Reconfigurable Array Architectures. *In : Handbook of Signal Processing Systems*, éd. par Bhattacharyya (S. S.), Deprettere (E. F.), Leupers (R.) et Takala (J.), pp. 449–484. – Springer US, janvier 2010.
14. Tehre (V.) et Kshirsagar (R.). – Survey on coarse grained reconfigurable architectures. *International Journal of Computer Applications*, vol. 48, n16, June 2012, pp. 1–7.
15. Yin (S.), Liu (D.), Liu (L.), Wei (S.) et Guo (Y.). – Joint affine transformation and loop pipelining for mapping nested loop on CGRAs. – In *DATE*, pp. 115–120, mars 2015.