Accelerating AI applications on a RISC-V processor

Adrien Pellé
IMT Atlantique
Brest, France
adrien.pelle@imt-atlantique.net

Le Nam Hieu Nguyen
IMT Atlantique
Brest, France
nam.nguyen-le@imt-atlantique.net

Rémi Vassal
IMT Atlantique
Brest, France
remi-raymond.vassal@imt-atlantique.net

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

L'accélération des réseaux de neurones convolutifs est devenue cruciale pour répondre aux exigences croissantes en termes de performances et d'efficacité énergétique dans le domaine de l'intelligence artificielle. Les réseaux de neurones convolutifs sont largement utilisés dans diverses applications telles que la vision par ordinateur, la reconnaissance vocale et la classification d'images. Cependant, leur exécution sur des architectures conventionnelles telles que les processeurs centralisés et les processeurs graphiques (GPU) peut être limitée en termes de vitesse et de consommation d'énergie.

Dans ce papier nous présentons notre méthode afin d'accélérer le temps d'exécution d'un réseau de neurones reconnaissant des chiffres de la base de données MNIST. Nos améliorations ont conduit à une accélération de plus de 300% d'un point de vue du nombre de cycles d'horloge nécessaire.

II. Background and related work

A. CVA6

CVA6 est un processeur compatible avec RISC-V qui peut être configuré en 32 ou 64 bits : CV32A6 et CV64A6.

CVA6 peut être configuré selon les besoins des utilisateurs et des applications grâce à plusieurs paramètres et fonctionnalités optionnelles (MMU, PMP, FPU, organisation et taille du cache...). Il vise les technologies FPGA et ASIC.

En tant que processeur, CVA6 peut exécuter de nombreux systèmes d'exploitation. Cela a déjà été démontré avec des distributions Linux embarquées (construites avec BuildRoot et Yocto), FreeRTOS et Zephyr. CVA6 dispose de l'interface de coprocesseur CV-X-IF pour étendre l'ensemble des instructions qu'il peut exécuter.

L'objectif de CVA6 est d'être pleinement conforme aux spécifications de RISC-V et de ne comporter aucune ou extrêmement peu d'extensions personnalisées (à l'exception des extensions sur l'interface CV-X-IF). CV32A6 et CV64A6 partagent le même code source SystemVerilog, disponible dans ce dépôt GitHub [**?**].

Identify applicable funding agency here. If none, delete this.

CV64A6 est une évolution industrielle d'ARIANE créée par l'ETH Zürich et l'Université de Bologne. CV32A6 est un ajout ultérieur de Thales. CVA6 est désormais géré par l'OpenHW Group par le biais de ses membres.

B. MNIST Application

L'application implémentée dans le processeur CVA6 est un réseau de neurones convolutif ayant pour objectif de reconnaître les chiffres de la base de donnée MNIST. Cette base regroupe un grand nombre de chiffre écrit à la main et en noir et blanc. Chacune des images a une taille de 28 pixels de coté.

Sans modification, le réseau de neurones est capable de faire une prédiction en environ 2,3 millions de cycles d'horloge.

C. Packed SIMD "P" Extension

L'extension "P" (Packed SIMD), est une extension du jeu d'instruction RISC-V, permettant d'effectuer des opérations SIMD à partir des general purpose registers. Cette extension est encore au stade de la proposition et n'est pas officielement ratifiée dans la norme RISC-V, contrairement à l'extension Vectorielle "V". Une implémentatation de l'extension "P" à été proposée dans la thèse de Davy Koene, de TU Delft [1]. L'architecture proposée dans cette thèse supporte une grande partie des instructions de l'extension "P" pour RV64.

De plus, des implémentations de la toolchain supportant l'extension P son disponible sur le Github de plctlab [2].

Notre approche à été dans un premier temps de reprendre l'architecture proposée dans la thèse de Davy Koene, en nous concentrant sur les instructions qui permettraient d'accélérer l'application. Afin d'utiliser les instructions supportées par notre nouvelle architecture, il faut modifier la toolchain afin qu'elle puisse supporter l'extension "P", nous utiliserons alors un fork de la toolchain développée par plctlab (Binutils [2] et GCC [3]). Par ailleurs, avec cette approche, on peut ajouter des instructions cusotmisées, en plus des instructions de l'extension P.

III. Architecture

A. Issue Stage

B. Execution Stage

La structure de CVA6 est composée de 5 étages : l'étage de frontend, l'étage de décodeur d'instructions, l'étage d'issue l'étape d'exécution et l'étape de commit. Dans le cadre de notre travail, nous utilisons principalement l'instruction

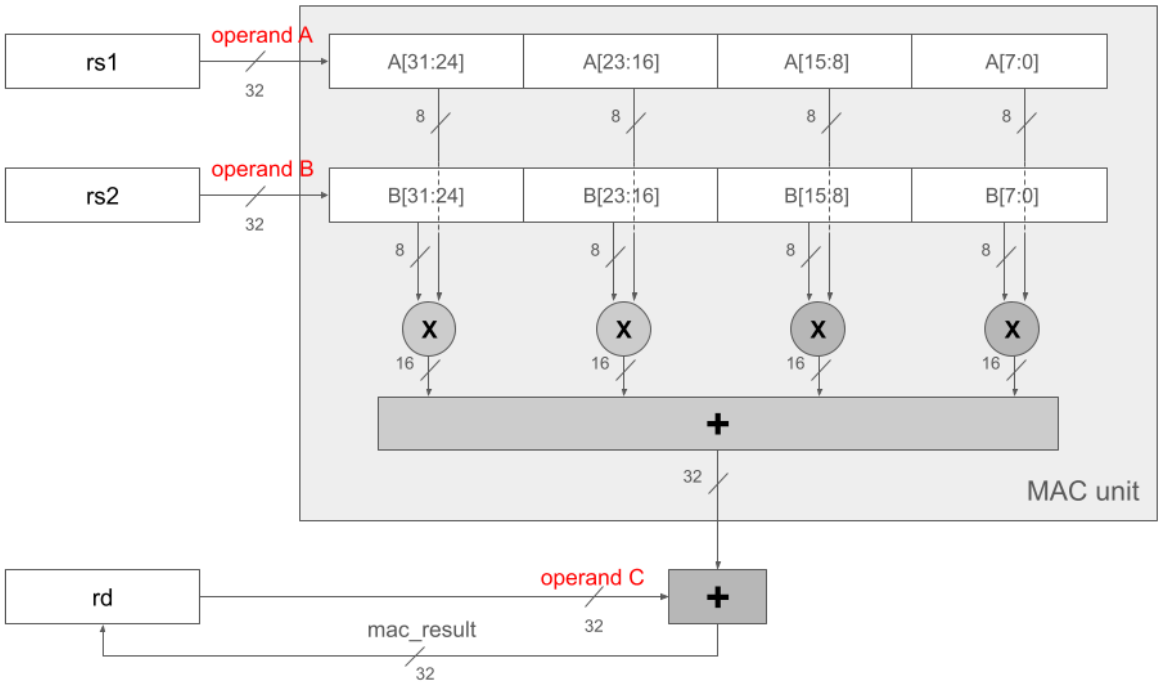

Fig. 1. Instruction SMAQA

SMAQA (Fig.1), qui nous permet d'effectuer plusieurs opérations d'addition, de multiplication et d'accumulation en même temps, ce qui entraîne une augmentation du parallélisme des données. et la réduction des horloges requises. Afin d'implémenter SMAQA, nous devons modifier 3 étages du CVA6 : décodeur d'instructions, issue et exécution. Alors que le décodeur d'instructions n'est que marginalement ajusté pour reconnaître les nouvelles instructions dans le paquet SIMD et que l'étage d'issue est discutée dans la section III.C, l'étage d'exécution est clarifiée dans différentes sections car elle subit le plus de modifications. En particulier, l'exécution contient des unités spécialisées qui sont responsables de leurs propres fonctionnalités telles que l'addition ou la multiplication. Ces unités sont directement affectées à l'exécution des calculs, ce qui signifie que ces pièces doivent être prises en compte en premier dans le cas d'une accélération des performances. Concrètement, les modifications architecturales de l'étape d'exécution se sont concentrées principalement sur l'Unité Arithmétique et Logique, spécialisée dans les additions, et sur le multiplicateur, spécialisé dans les multiplications et l'accumulation.

C. Multiplier

Dans l'application MNIST, les réseaux de neurones sont essentiellement les poids. Chaque poids est multiplié par son entrée correspondante (un pixel de l'image ou une sortie d'une couche précédente), puis les produits sont accumulés pour former une sortie. Par conséquent, le SMAQA dans SIMD qui permet d'effectuer simultanément une opération d'addition-multiplication avec de nombreuses entrées et poids est idéal pour améliorer les calculs. Selon le package SIMD et l'architecture de CVA6, cette opération doit être implémentée dans l'unité multiplicatrice.

Concernant le type de données, les entrées sont des entiers non signés 8 bits, les poids sont des entiers signés 8 bits, le SMAQA effectue donc les calculs sous forme de signe étendu pour les poids (au lieu de zéro-étendu).

Dans notre premier effort, l'instruction SMAQA à 32 bits est implémentée, ce qui signifie que le calcul a l'opérande de 32 bits. Chaque poids et entrée sont de 8 bits, donc 4 paires d'entrées et de poids sont utilisées en même temps. L'opération SMAQA est visible sur la Fig. 1, 4 éléments de 8 bits sont enregistrés dans le GPR (Registre à usage général de longueur 32 bits), et l'instruction SMAQA prend trois opérandes en entrées : opérande A (4×8 bits), opérande B (4×8 bits) et opérande C (32 bits). Tout d'abord, chaque opérande est stocké dans un registre, puis une multiplication non signée est effectuée entre chaque élément de 8 bits. Ensuite, les résultats de chaque multiplication sont additionnés. Enfin, ils sont ajoutés à l'opérande C, et le résultat final est stocké dans le propre registre de l'opérande C, appelé registre de destination ou accumulateur. Initialement, il n'y a que 2 ports de lecture sur le fichier de registre, alors que l'instruction SMAQA utilise 3 entrées qui nécessitent un port de lecture supplémentaire pour pouvoir lire l'opérande C depuis le registre de destination, il faut donc ajouter un port de lecture au fichier de registre. L'adresse de l'opérande C est envoyée par l'étape d'émission à l'étape d'exécution, c'est la valeur lue dans le champ "rd" de l'instruction SMAQA.

Dans notre prochain effort, basant l'idée de SMAQA avec 32 bits, SMAQA avec 64 bits sont implémentés pour améliorer

avons alors désactivé les patterns des instructions non désirées, en modifiant les fichiers du compilateur gcc.

B. P extension instructions

Instructions ”P” ajoutées :

C. Custom instructions

Instructions custom ajoutées :

V. Results

A. Methodology

Nous présentons les résulats pour différentes implémentations. Tout d'abord, les résultats de l'application MNIST de référence seront présentées. Puis, les résultats avec l'instruction SMAQA (sur des éléments de 32 bits, avec 3 ports de lecture au registerfile). Enfin, les resultats avec l'instruction SMAQA64 (5 ports de lectures sur le register file).

Nous comparons les performances de chaque fonction du réseau de neurone :

- CONV 1 : Première couche de convolution
- CONV 2 : Seconde couche de convolution
- FC1 : Première couche fully connected
- FC2 : Seconde couche fully connected

B. Performances

Table I

C. Ressources utilisation

VI. Conclusions

References

[1] Koene, Davy. ”Implementation and evaluation of packed-simd instructions for a risc-v processor.” (2021).
[2] https://github.com/plctlab/riscv-binutils-gdb/tree/riscv-binutils-p-ext
[3] https://github.com/plctlab/riscv-gcc/tree/riscv-gcc-p-ext
[4] https://github.com/openhwgroup/cva6/

TABLE I
COMMPARAISON DE PERFORMANCES ENTRE DIFFÉRENTES ARCHITECTURES DE CVA6 POUR L'APPLICATION MNIST, SUR L'EVAL BOARD ZYBO-Z7

Instance		MNIST(Golden)	SIMD MNIST(SMAQA32)	SIMD MNIST(SMAQA64)
Nombre de ports de lecture du Register File		2	3	5
CONV1	Nombre d'instructions	207 537	70 683	70507
	Nombre de cycles d'horloge	296 874	98 385	98170
	Amélioration	100%	302%	302%
CONV2	Nombre d'instructions	1 098 263	253 463	166 692
	Nombre de cycles d'horloge	1 468 114	435 107	294821
	Amélioration	100%	337%	498%
FC1	Nombre d'instructions	408 170	91 518	53118
	Nombre de cycles d'horloge	559 462	174 671	111 199
	Amélioration	100%	320%	503%
FC2	Nombre d'instructions	10 656	10 656	10 656
	Nombre de cycles d'horloge	14 804	14 865	14826
	Amélioration	100%	100%	100%
Autre	Nombre d'instructions	432	-	-
	Nombre de cycles d'horloge	67	-	-
	Amélioration	100 %	- %	- %
Total	Nombre d'instructions	1 731 593	426 385	301789
	Nombre de cycles d'horloge	**2 354 247**	**723 235**	**527 913**
	Amélioration	**100%**	**326%**	**446%**