

Étude de plusieurs architecture pour effectuer divers opérations vectorielles

Multiplication vectorielle

Dans cette partie, nous réalisons la conception et l'étude d'un multiplieur vectoriel, i.e faisant le produit:

$$u * v = (u_1 * v_1, u_2 * v_2, \dots, u_{Ndata} * v_{Ndata})$$

Cela par la réalisation des fichiers mul.v et multiply.v et l'utilisation des testbenchs -

- tb_multiply2.v
- tb_multiplyNmul.v
- tb_multiply_seq.v
- tb_multiply_unitary.v
- tb_multiply.v

Combien de cycles le produit prend il ? (tb_multiply_seq)

La période entre chaque cycles est de 4 secondes, donc la fréquence est de 0.25 et le calcul démarre à partir de 40 secondes.

Or la valeur de out est stable à partir du cycle qui finit à 77s donc on soustrait pour obtenir une durée de 37 secondes pour le calcul.

Cela nous donne ainsi $37 \times 0.25 \approx 9$ cycles pour effectuer complètement la multiplication vectorielle.

Combien de cycles le produit prend il ? (tb_multiply2)

Avec GTKWAVE, on peut observer qu'il y a la bonne valeur dans out à la toute fin, donc le multiplier2 est fonctionnel.

La valeur de out est stable à partir de 53 secondes, donc la multiplication a duré 13 secondes.

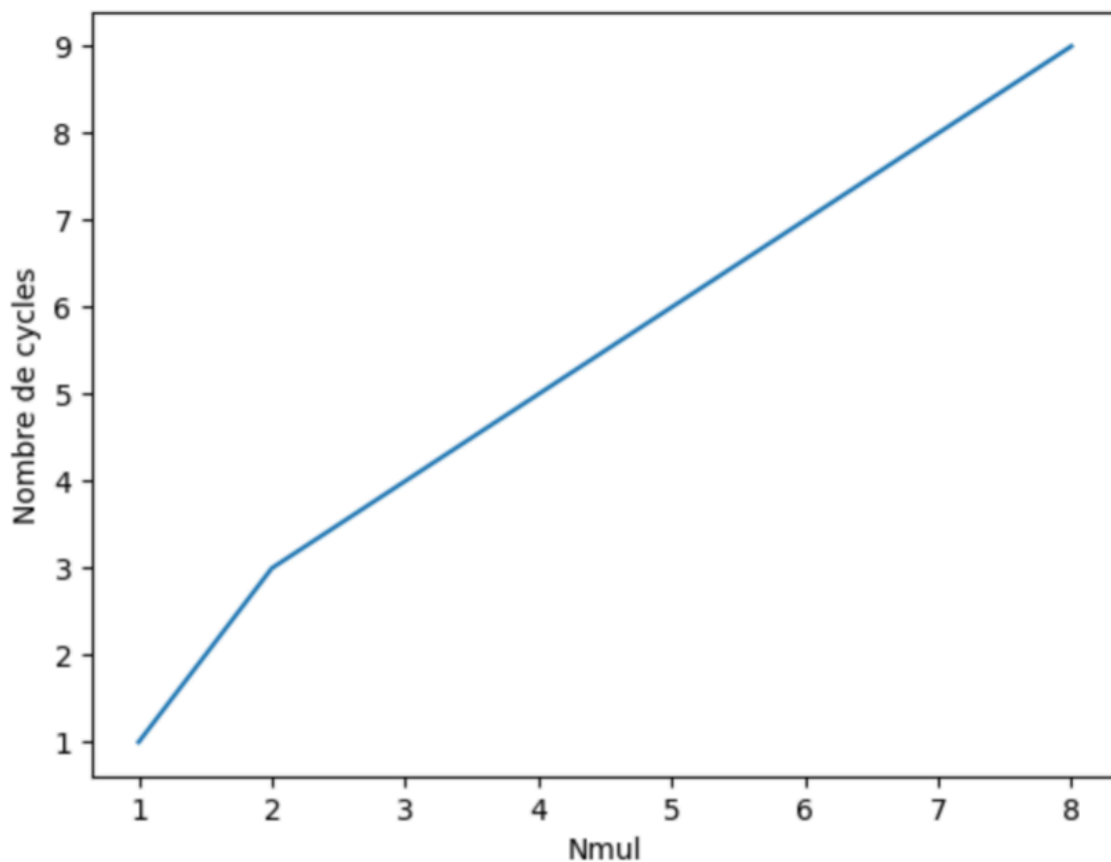
Par conséquent, le nombre de cycles nécessaires est de $13 \times 0.25 \approx 3$ cycles.

Étude du nombre de cycles avec Nmul multiplieurs en parallèle

Nmul est un diviseur de Ndata

Nmul	Durée en secondes	Nombre de cycles
1	5	1
2	13	3

Nmul	Durée en secondes	Nombre de cycles
3	17	4
4	21	5
5	25	6
6	29	7
7	33	8
8	37	9



Que se passerait-il si Nmul n'était pas un diviseur de Ndata ?

Nmul	Durée en secondes	Nombre de cycles	Ndata/Nmul	Nombre d'instance de Mul
1	5	1	8	8
2	13	3	4	4
3	17	4	2.6	2
4	21	5	2	2
5	25	6	1.6	1
6	29	7	1.33	1

Nmul	Durée en secondes	Nombre de cycles	Ndata/Nmul	Nombre d'instance de Mul
7	33	8	1.14	1
8	37	9	1	1

Si Nmul n'est pas un diviseur de Ndata, alors le nombre de multiplieurs élémentaires instanciés sera égal à la partie entière inférieure de $Ndata/Nmul$

En revanche si $Nmul > Ndata$, on aura cette erreur

```
tb_multiplyNmul.v:34: error: part select A_tmp[-1:0] is out of order.
tb_multiplyNmul.v:35: error: part select B_tmp[-1:0] is out of order.
2 error(s) during elaboration.
```

Produit Scalaire

Produit scalaire

Le produit scalaire de 2 vecteurs \vec{A} et \vec{B} de n éléments se calcule par la formule :

$$\vec{A} \cdot \vec{B} = \sum_{i=1}^{i=n} a_i \times b_i$$

où a_i et b_i sont les coordonnées des vecteurs \vec{A} et \vec{B} .

Il y a deux façons de mettre en œuvre le produit scalaire :

- en sommant les composantes résultant de la multiplication vectorielle vu précédemment,
- en utilisant un opératuer de multiplication-accumulation (MAC).

Architecture avec multiplication vectorielle

Nous avons calculer précédemment le nombre de cycles nécessaires pour la multiplication vectorielle en fonction du nombre de multiplicateur élémentaire, notons-cela $cycles(MV_{Nmul})$.

Une fois la multiplication vectoriel effectué, nous faisons une somme sur le vecteur résultant, et cela prend 1 cycle.

Le nombre de cycles en fonction du nombre de multiplieurs élémentaires est donc donné par

$$cycles(MV_{Nmul}) + 1$$

Architecture avec MAC

Le testbench du scalar_product_mac fait dans un premier temps le produit scalaire

$\langle (1, 2, 3, 2), (6, 5, 4, 1) \rangle$ dont la valeur est 30.

Ce produit scalaire commence à partir de 40ns pour finir à 58ns et prend 4 cycles pour que la valeur dans la sortie soit bien à 30.

Cela peut se voir facilement avec gtkwave ou bien par calcul $18 \times 0.25 \approx 4$.

Nous pouvons conjecturer que pour des vecteurs de longueur $Ndata$, ce circuit prendra $Ndata + 1$ cycles pour effectuer le produit scalaire entre les deux.

Utilisation de plusieurs MAC en parallèle

Pour effectuer le produit scalaire en parallèle avec des produits scalaires utilisant un seul MAC, il faut diviser le vecteur en $Nmac$ morceaux de longueurs $Ndata/Nmac$

En effet, avec $m = Ndata$ et $N = Nmac = 4$,

$$\sum_{i=1}^m a_i b_i = \underbrace{\sum_{i=1}^{m/4} a_i b_i}_{SPMAC_1} + \underbrace{\sum_{i=m/4+1}^{2m/4} a_i b_i}_{SPMAC_2} + \underbrace{\sum_{i=2m/4+1}^{3m/4} a_i b_i}_{SPMAC_3} + \underbrace{\sum_{i=3m/4+1}^m a_i b_i}_{SPMAC_4}$$

Et donc $\forall N \mid m$,

$$\sum_{i=1}^m a_i b_i = \sum_{k=1}^N \underbrace{\sum_{(k-1)\frac{m}{N}+1 \leq i \leq k\frac{m}{N}} a_i b_i}_{SPMAC_k}$$

Chaque MAC fera son produit scalaire en $Ndata$ cycles et l'opération de réduction est faite par un circuit combinatoire et qui se fait en 1 cycle.

Par conséquent, le circuit avec $Nmac$ MAC parallèles prendra $Ndata/Nmac + 1$ cycles pour faire le produit scalaire de deux vecteurs de longueurs $Ndata$.

À noter que dans le test_bench, out fait 8 bits mais on test sur deux vecteurs dont le produit scalaire est sur plus que 8 bits, je l'ai donc changé pour le remplacer par le produit scalaire

$$< (10, 10, 1, 5), (10, 10, 5, 1) > = 210$$

Après reset, le circuit nous donne bien la bonne valeur.

Récapitulatif

Tableau comparatif des différentes architectures pour réaliser un produit scalaire entre deux vecteurs de $Ndata$ éléments.

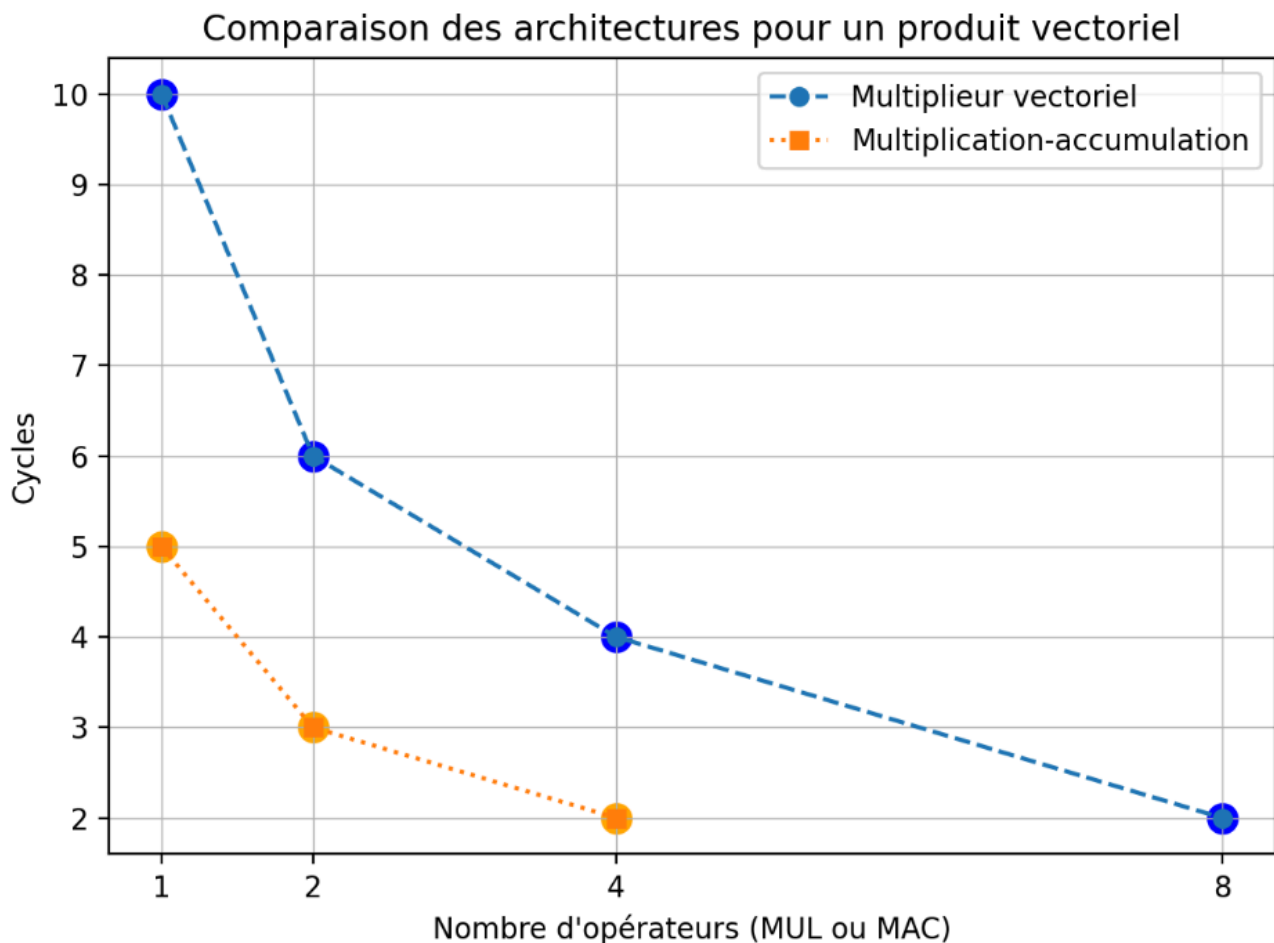
On distingue 3 architectures: une à base de multiplieurs élémentaires en parallèle, une à base de mac en parallèle, et celle utilisant une pipeline.

Pour chaque architecture, on note son nombre de cycles en fonction du nombre d'opérateurs élémentaires qu'elle utilise (MUL ou MAC).

Et on pose $Ndata = 4$,

# Opérateurs	Multiplieur vectoriel	MAC	Pipeline
1	10	5	-

# Opérateurs	Multiplieur vectoriel	MAC	Pipeline
2	6	3	-
4	4	2	-
8	2	-	8



Produit matriciel

Dans cette partie, nous nous concentrons sur la réalisation et l'étude de plusieurs architectures pour effectuer le produit matriciel

$$Ax = y$$

Comparaison des architectures

Soit A une matrice de M lignes et N colonnes.

Le produit scalaire combinatoire prend Ndata cycles, celui avec mac prend Ndata + 1 cycle et celui avec pipeline prend Ndata cycles.

On a Mdata produit scalaire en parallèle, donc on obtient les valeurs suivantes:

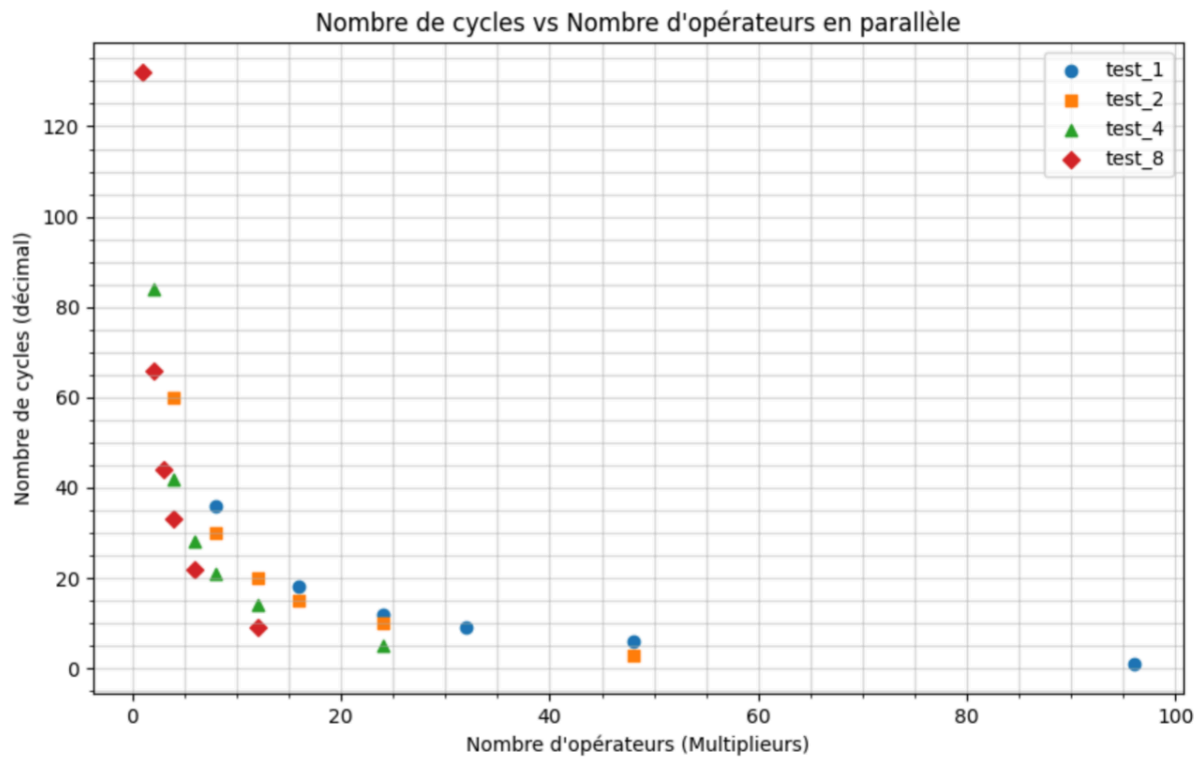
Architecture	#Opérateurs élémentaires (multiplieur ou MAC)	#Cycles
comb	8	4

Architecture	#Opérateurs élémentaires (multiplieur ou MAC)	#Cycles
mac	4	5
pipe	4 étages	8

Analyse des résultats

test_i_j: série *i* avec une multiplication matricielle utilisant *j* produits scalaires

fichier	#multiplieurs	cycles (dec)
test_1_1	8	36
test_1_2	16	18
test_1_3	24	12
test_1_4	32	9
test_1_6	48	6
test_1_12	96	1
test_2_1	4	60
test_2_2	8	30
test_2_3	12	20
test_2_4	16	15
test_2_6	24	10
test_2_12	48	3
test_4_1	2	84
test_4_2	4	42
test_4_3	6	28
test_4_4	8	21
test_4_6	12	14
test_4_12	24	5
test_8_1	1	132
test_8_2	2	66
test_8_3	3	44
test_8_4	4	33
test_8_6	6	22
test_8_12	12	9



Le choix d'une architecture par rapport à une autre va dépendre de nos contraintes de performances et de surfaces.

Pour cela, on va choisir un point dit "pareto-optimal", i.e qui n'est pas pareto-dominé par aucune autre architecture (aucun meilleur point).

Or il y en a plusieurs, donc en fonction de si on privilégie la surface ou la performance, il faudra faire un choix.

Par exemple si une contrainte impose que ça ne doit pas dépasser 18 cycles et que le coût en multiplieurs ne doit pas dépasser 20, alors on va choisir la configuration de la série 8 avec 12 multiplieurs pour notre architecture.