Rapport projet SIMD

NGUYEN jean-Baptiste

Contents

1 Introduction

Ce projet vise à comparer les performances des multiplications de matrice en utilisant les instructions AVX. Les tests ont été fait avec des types int32 float et double en comparant les nombre de cycles effectué entre une version utilisant les vecteur AVX512 et une version séquentiel.

2 Usage

Le projet contient un dossier src qui contient tous les sources servant à creer la bibliothèque statique. Le dossier test contient les programmes servant à calculer le nombre de cycle pour chaque type. Pour tout compiler il suffit d'exécuter la commande make à la base du projet. Enfin chaque programme prend 3 parametre: le nombre d'itération, le nombre d'échantillon, et la taille de la matrice.

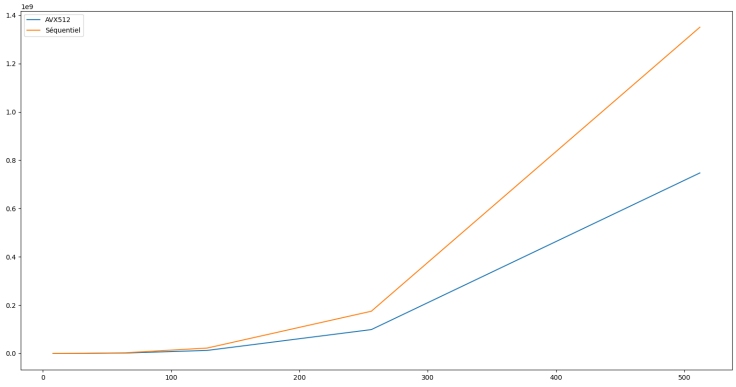

Figure 1: courbe des performances pour la mutliplication des int

3 Implémentation

J'ai fait le choix de représenter les matrices par des tableaux à une dimension.

3.1 matrice into vecteur

Cette fonction sert à représenter une matrice en pointeur sur vecteur SIMD, le but de cette fonction est de représenter chaque ligne de la matrice de départ par un ou plusieur vecteur SIMD qui sont ensuite placé les un à la suite des autre.

3.2 int32

Pour la version int32 j'ai fait le choix de n'en placer que 8 dans les vecteur m512i là où on pourrait en stocker 16 normalement, je les traite comme si c'était en réalité des int64. Cela avait pour but d'éviter l'overflow lors des différente multiplication ce qui ne marche pas forcément car le programme effectue des sommes de multiplication de int32 ce qui a une chance de dépasser les 64 bit.

4 Résultat

Le facteur d'accélération augmente avec la taille des matrices jusqu'à 1.8 pour les int 1.6 pour les float et 1.5 pour les doubles. Je n'ai pas réussi à atteindre la barre des 2 pour le facteur malgrès le fait que pour les float dans chaque vecteur m512, 16 y sont stocké. Peut être que la portion du code parallélisé grâce aux instructions AVX est trop petite par rapport à l'intégralité de celui-ci.

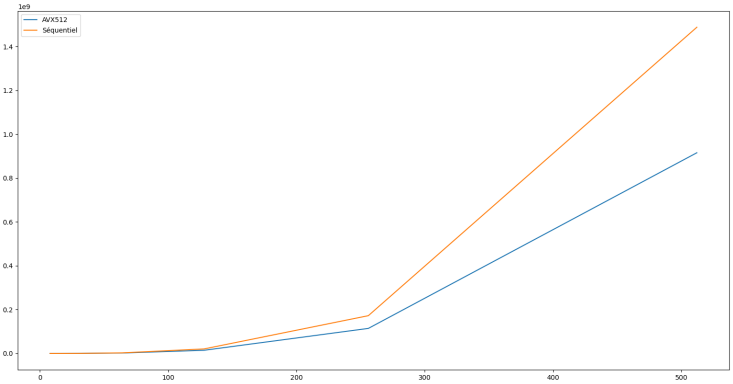

Figure 2: courbe des performances pour la mutliplication des float

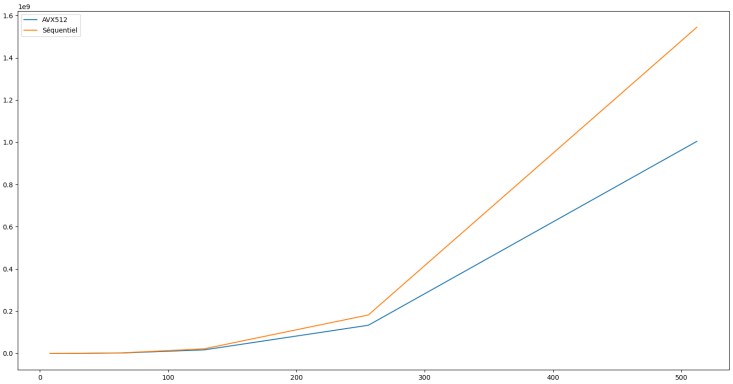

Figure 3: courbe des performances pour la mutliplication des double

Taille n	**AVX512 (cycles)**	**Séquentiel (cycles)**	**Accélération**
8	3 635	5 423	1.49
16	26 127	41 597	1.59
32	209 311	385 569	1.84
64	1 574 566	2 753 013	1.75
128	12 388 152	22 189 556	1.79
256	98 690 650	174 698 526	1.77
512	747 216 551	1 350 404 217	1.81

Table 1: Comparaison des performances pour la multiplication de matrices `int`

Taille n	**AVX512 (cycles)**	**Séquentiel (cycles)**	**Accélération**
8	8 501	5 719	0.67
16	31 404	39 630	1.26
32	240 488	322 280	1.34
64	1 827 327	2 579 517	1.41
128	14 391 836	20 563 445	1.43
256	114 136 199	172 177 326	1.51
512	915 333 265	1 487 873 847	1.63

Table 2: Comparaison des performances pour la multiplication de matrices `float`

Taille n	**AVX512 (cycles)**	**Séquentiel (cycles)**	**Accélération**
8	4 764	5 354	1.12
16	34 655	41 006	1.18
32	272 833	337 501	1.24
64	2 145 871	2 762 474	1.29
128	16 799 021	21 822 659	1.30
256	133 338 097	182 100 248	1.37
512	1 004 035 223	1 544 449 524	1.54

Table 3: Comparaison des performances pour la multiplication de matrices `double`

5 Perspective

Pour faire le produit scalaire entre deux vecteur SIMD j'utilise d'abord une instruction AVX pour faire toutes les multiplications puis je somme les résultat séquentiellement. Une solution pour accélérer le programme serait d'éssayer de paralléliser cette somme à chaque produit scalaire.