

Optimisation SIMD pour le calcul régulier

Lionel Lacassagne

Introduction

Le but de ce TP est de manipuler les instructions SIMD flottantes pour implémenter les algorithmes de base en algèbre linéaire (BLAS = *Basic Linear Algebra System*): le produit scalaire (produit vecteur-vecteur) et le produit matricielle (produit matrice-matrice) Les fichiers C nécessaires se trouvent à l'adresse www.ief.u-psud.fr/~lacas/Teaching.

Pour chaque codage, donnez, dans un tableau récapitulatif, les spécificités de votre code, à savoir la complexité arithmétique (MUL+ADD), le nombre d'accès mémoire (LOAD+STORE) et l'intensité arithmétique (le ratio entre la complexité arithmétique et le nombre d'accès mémoire). Si nécessaire, la documentation du compilateur Intel (options de compilation) est disponible sur: <http://www.ief.u-psud.fr/~lacas/Download/Download.html>

1 Opérateurs 1D

1.1 Addition de deux vecteurs (tableaux 1D) de registres SIMD

Soient les vecteurs \vec{X}_1 et \vec{X}_2 de taille n et \vec{Y} leur somme:

$$\vec{Y} = \vec{X}_1 + \vec{X}_2 = \sum_{i=0}^{n-1} X_1(i) + X_2(i) \quad (1)$$

- Codez cette addition SIMD dans la fonction `add_vf32vector()`. La fonction de test (réalisant les allocations mémoire) est `test_add_dot_vf32vector()`
Afin de séparer les instructions de calcul des instructions d'accès mémoire (principe des processeurs RISC), utilisez les registres SIMD x_1 , x_2 et y tels que $x_1 = X_1(i)$, $x_2 = X_2(i)$ et $y = Y(i)$.

1.2 Produit scalaire de deux vecteurs (tableaux 1D) de registres SIMD: réduction totale

Soient les vecteurs \vec{X} et \vec{Y} de taille n et d leur produit:

$$d = \vec{X}_1 \cdot \vec{X}_2 = \sum_{i=0}^{n-1} X_1(i) \times X_2(i) \quad (2)$$

- Codez ce produit scalaire SIMD dans la fonction `dot_vf32vector`. La fonction de test et d'allocation mémoire est `test_add_dot_vf32vector`. Que faut il faire en fin de traitement pour que le produit scalaire complet (la somme de **tous** les produits) soit dans chacun des quatre blocs du registre SIMD ?

1.3 Somme sur un voisinage de 3 points (tableaux 1D) de registres SIMD: réduction partielle

Les opérateurs précédents réalisent des calculs entre registres SIMD. Une difficulté du calcul SIMD est la réalisation de calcul à l'intérieur d'un registre SIMD.

Soit le tableau X . On souhaite que le tableau Y contienne en tout point la somme de trois points du tableau X :

$$Y_3(i) = X(i - 1) + X(i) + X(i + 1) \quad (3)$$

1. Afin de simplifier le problème, on ne calculera pas, ni $Y(0)$, ni $Y(n - 1)$.
2. Codez cet opérateur dans la fonction `sum3_vf32vector`. La fonction de test et d'allocation est `test_sum_vf32vector`.
3. Validez votre code en comparant les résultats avec un tableur.

2 Produit matriciel

Soient A , B et C trois matrices carrées de taille $n \times n$. Pour toutes les versions (scalaires et SIMD) vous donnerez les performances en *cpp* et MFlops, pour des tailles de matrices $n = 100$, $n = 500$, $n = 1000$. Lire les explications ci-dessous avant de commencer à coder.

1. Coder les versions scalaires `ijk` et `ikj` dans les fonctions `mul_ijk` et `mul_ikj`.
2. Coder la méthode classique de multiplication par blocs de taille 4×4 avec transposition (produit ligne \times colonne, voir explications ci-dessous). Valider le code en comparant le résultat avec le même calcul réalisé dans un tableur (Excel, Calc ou Numbers).
3. Coder les versions SIMD `mul_ijk_SSE` et `mul_ikj_SSE` qui utiliseront la méthode classique de multiplication par blocs.
4. Coder la méthode évoluée de multiplication par blocs de taille 4×4 sans transposition. Valider le code en comparant le résultat avec le même calcul réalisé dans un tableur (Excel, Calc ou Numbers).
5. Coder les versions SIMD `mul_ijk_SSE_splat` et `mul_ikj_SSE_splat` qui utiliseront la méthode évoluée de multiplication par blocs.

3 Multiplication par blocs 4×4

Soient 3 matrices A_4 , B_4 , C_4 de taille 4×4 :

$$A_4 = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{40} & a_{31} & a_{32} & a_{33} \end{bmatrix}, B_4 = \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{40} & b_{31} & b_{32} & b_{33} \end{bmatrix}, C_4 = \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{40} & c_{31} & c_{32} & c_{33} \end{bmatrix} \quad (4)$$

Soient $\vec{a}_0, \vec{a}_1, \vec{a}_2, \vec{a}_3$ les 4 vecteurs **ligne** de A et soient $\vec{b}_0, \vec{b}_1, \vec{b}_2, \vec{b}_3$ les 4 vecteurs **colonne** de B_4 . Soient \vec{c}_0 le premier vecteur ligne de C_4 :

$$\begin{aligned}
\vec{a}_0 &= [a_{00} & a_{01} & a_{02} & a_{03}] \\
\vec{a}_1 &= [a_{10} & a_{11} & a_{12} & a_{13}] \\
\vec{a}_2 &= [a_{20} & a_{21} & a_{22} & a_{23}] \\
\vec{a}_3 &= [a_{30} & a_{31} & a_{32} & a_{33}] \\
\vec{b}_0 &= [b_{00} & b_{10} & b_{20} & b_{30}] \\
\vec{b}_1 &= [b_{01} & b_{11} & b_{21} & b_{31}] \\
\vec{b}_2 &= [b_{02} & b_{12} & b_{22} & b_{32}] \\
\vec{b}_3 &= [b_{03} & b_{13} & b_{23} & b_{33}] \\
\vec{c}_0 &= [c_{00} & c_{10} & c_{20} & c_{30}]
\end{aligned}$$

Soient b_0, b_1, b_2, b_3 les 4 registres SIMD ligne de B_4 :

$$\begin{aligned}
b_0 &= [b_{00} & b_{01} & b_{02} & b_{03}] \\
b_1 &= [b_{10} & b_{11} & b_{12} & b_{13}] \\
b_2 &= [b_{20} & b_{21} & b_{22} & b_{23}] \\
b_3 &= [b_{30} & b_{31} & b_{32} & b_{33}]
\end{aligned}$$

Il existe (au moins) deux implémentations SIMD de la multiplication matricielle par blocs 4×4 .

Algorithm 1: standard 4×4 vector-matrix multiplication with transposition (where \times is `_mm_mul_ps` and $+$ is `_mm_add_ps`)

Input: A and B , two 4×4 matrix

Output: C one 4×4 matrix

- 1 load \vec{a}_0 from A_4
 - 2 load b_0, b_1, b_2, b_3 from B_4
 - 3 transpose b_0, b_1, b_2, b_3 to get $\vec{b}_0, \vec{b}_1, \vec{b}_2, \vec{b}_3$
 - 4 compute $u_0 = a_0 \times b_0, u_1 = a_0 \times b_1, u_2 = a_0 \times b_2, u_3 = a_0 \times b_3$
 - 5 accumulate $c_0 = u_0 + u_1 + u_2 + u_3 = [\vec{a}_0 \cdot \vec{b}_0, \vec{a}_0 \cdot \vec{b}_1, \vec{a}_0 \cdot \vec{b}_2, \vec{a}_0 \cdot \vec{b}_3]$
-

Le premier algorithme (algo. 1) est basé sur le classique produit scalaire “*ligne \times colonne*” : afin de pouvoir réaliser la multiplication point-à-point de \vec{a} par \vec{b} , il faut transposer \vec{b} . Comme en SIMD la transposition impliquer autant de registres SIMD qu’il y a de blocs à l’intérieur d’un registre, il faut transposer 4 registres puis réaliser les multiplications point à point pour obtenir les 4×4 produits partiels dans 4 registres SIMD. Après cela, il faut encore transposer ces 4 registres avant de pouvoir faire les accumulation et enfin obtenir 4 produits scalaires d’un vecteur de A_D par 4 vecteurs de B_4 .

Le second algorithme, plus rapide (algo. 3) consiste à “*splater*” (algo. 2) chaque registre de A dans 4 registres temporaires s (s_0, s_1, s_2, s_3). Il suffit alors d’accumuler les 4 produits partiels comme dans le cas précédent pour obtenir 4 produits scalaires. Ainsi, il n’y a plus de transposition de vecteurs de B_4 ni de transposition entre les phases de multiplication et d’accumulation.

Algorithm 2: splat = broadcast block # i of v to all blocks of w:

Input: v an SIMD register, i an index

Output: w an SIMD register

- 1 broadcast block # i of v to all blocks of w
 - 2 $n \leftarrow \text{sizeof}(v)$ [n=2 for double, n=4 for float, 8 for 16-bit integer, 16 for 8-bit integer]
 - 3 **for** $k = 0$ **to** $n - 1$ **do**
 - 4 $w[k] \leftarrow v[i]$
-

Algorithm 3: fast 4×4 vector-matrix multiplication based on vector splat (where \times is `_mm_mul_ps` and $+$ is `_mm_add_ps`)

Input: A_4 and B_4 , two 4×4 matrix

Output: C_4 , one 4×4 matrix

- 1 load a_0 from A_4
 - 2 splat all blocks of a_0 into 4 SIMD registers:
 $s_0 \leftarrow \text{splat}(a_0, 0), s_1 \leftarrow \text{splat}(a_0, 1), s_2 \leftarrow \text{splat}(a_0, 2), s_3 \leftarrow \text{splat}(a_0, 3)$
 - 3 load b_0, b_1, b_2, b_3 from B_4
 - 4 transpose b_0, b_1, b_2, b_3 to get $\vec{b}_0, \vec{b}_1, \vec{b}_2, \vec{b}_3$
 - 5 compute $u_0 = s_0 \times b_0, u_1 = s_1 \times b_1, u_2 = s_2 \times b_2, u_3 = s_3 \times b_3$
 - 6 accumulate $c_0 = u_0 + u_1 + u_2 + u_3 = [\vec{a}_0 \cdot \vec{b}_0, \vec{a}_0 \cdot \vec{b}_1, \vec{a}_0 \cdot \vec{b}_2, \vec{a}_0 \cdot \vec{b}_3]$
-

4 Application numérique

$$A_4 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}, B_4 = \begin{bmatrix} 10 & 11 & 12 & 13 \\ 20 & 21 & 22 & 23 \\ 30 & 31 & 32 & 33 \\ 40 & 41 & 42 & 43 \end{bmatrix}, C_4 = A_4 \times B_4 = \begin{bmatrix} 300 & 310 & 320 & 330 \\ 700 & 726 & 752 & 778 \\ 1100 & 1142 & 1184 & 1226 \\ 1500 & 1558 & 1616 & 1674 \end{bmatrix}$$