Ecole Polytechnique de l'Université de Tours
64 Avenue Jean Portalis,
37200 Tours, France
Tel: +33 02 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
2013 - 2014

Rapport de Projet SE

Mise en œuvre d'une architecture parallèle SIMD / multi-cœur pour la comparaison binaire

Auteurs
LIU Fa
fa.liu@univ-tours.fr
ZHOU Haiying
haiying.zhou@univ-tours.fr

Encadrant:
Mathieu DELALANDRE
mthieu Delalandre@univ-tours.fr

Version du 9 juin 2014

Remerciements

M. Mathieu DELALANDRE

Résumé

Ce rapport est pour le projet « Mise en œuvre d'une architecture parallèle SIMD / multi-cœur pour la comparaison binaire - application à la protection de copyright de Manga ». Dans le rapport, on explique le principe de fonctionnement de SIMD et les instructions de SIMD. On aussi présente notre projet avec l'analyse et l'explication. Dans le projet, on exécute l'opération de XOR sur les données. Afin de montre l'avantage de SIMD, on utile la méthode de SIMD et la méthode général avec langage C++.

Mots clés : SIMD, XOR

Table des Matières

Liste des tableaux **1**
Table des figures **2**
1 Introduction 3
2 SIMD 3
2.1 C'est quoi le SIMD? 3
2.2 Le principe du SIMD 4
2.3 Instructions SIMD 5
2.4 Application 6
2.5 SSE 7
2.5.1 Le Streaming SIMD Extensions (SSE) 7
2.5.2 Instructions SSE 8
3 Mise en œuvre de SIMD: Projet SE 9
3.1 Opération bit a bit 9
3.2 La comparaison binaire par langage C 10
3.3 La comparaison binaire par SIMD 11
3.4 La distinction entre SIMD et SISD 13
4 conclusions 14
Bibliographie **15**

Liste des tableaux

Tableau1 la vérité A XOR B 9
Tableau2 l'opérateur binaire XOR de langage C 10
Tableau3 L'opérateur binaire XOR de SSE 10
Tableau4 La fonction «CompareDate» 10
Tableau5 La fonction «CompareDate_SSE» 12
Tableau6 La fonction «main» 13

Table des figures

Figure1 Single Instruction on Multiple Data. 3
Figure2 Le principe du mode SIMD. 4
Figure3 La taille des paquets. 5
Figure4 Les registres XMM des processeurs x86 7
Figure5 L'exécution d'une instruction d'addition vectorielle 7
Figure6 Les résultats d'exécution du projet. 13

1 Introduction

Le sujet de ce projet est Mise en œuvre d'une architecture parallèle SIMD / multi-cœur pour la comparaison binaire - application à la protection de copyright de Manga. L'objectif de ce projet est la mise en œuvre de ces instructions au sein d'une architecture parallèle multi-cœur (i.e. traitement sur plusieurs cœurs d'instruction SIMD). On pourra prendre comme cas d'usage la comparaison de « templates » (i.e. images) binaires pour la protection de copyright de Manga. Dans ce contexte, les instructions mises en œuvre seront principalement de type « bitwise operator ». On s'intéressera aux mécanismes mis en place au sein du système pour la protection des changements. Le projet est réalisé avec les instructions SIMD.

2 SIMD

2.1 C'est quoi le SIMD?

SIMD, Single Instruction on Multiple Data, peut copier plusieurs opérandes et les emballés dans un grand ensemble registre d'instructions. Les instructions SIMD sont capables de s'appliquer à plusieurs données en simultanées. Il utilise un contrôleur pour contrôler plusieurs processeurs, il effectue la mêmes opération sur un ensemble de données (également connue sous le nom "vecteur de données") en même temps, pour atteindre l'espace de la technologie de parallélisme. Ces instructions peuvent traiter des mots de dimensions 64 à 128 bits sur processeur, et de 128 à 256 bits sur GPU. Elles sont souvent couplées à des instructions vectorielles dites AVX (Advanced Vector Extensions) et FMA (Fused Multiply Add). Les opérations FMA fusionnent une multiplication et une addition, c'est-à-dire une opération de la forme A = A × B + C. La mise en place d'instructions SIMD implique un support particulier du système d'exploitation, sur les aspects changement et restauration de contexte [1].

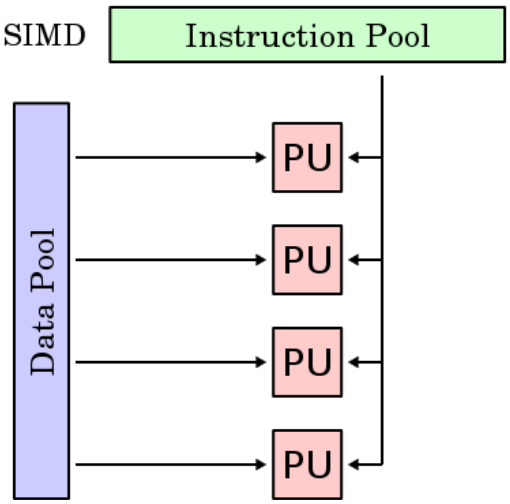

Figure1 Single Instruction on Multiple Data

SIMD est une des quatre catégories d'architecture définies par la taxinomie de Flynn en 1966 et désigne un mode de fonctionnement des ordinateurs dotés de plusieurs unités de calcul fonctionnant en parallèle. Dans ce mode, la même instruction est appliquée simultanément à plusieurs données pour produire plusieurs résultats.

Par opposition à SISD (Single Instruction on Single Data), le fonctionnement traditionnel, et MIMD

(Multiple Instructions on Multiple Data), le fonctionnement avec plusieurs processeurs aux mémoires indépendantes, SIMD a l'avantage sur performance. Par exemple, dans l'instruction d'addition, après le décodage des instructions, d'abord l'unité d'exécution du SISD va accéder à la mémoire, obtenir le premier opérande, puis elle va accéder à la mémoire encore, obtenir le second opérande, et puis à effectuer la sommation. Et dans la CPU de SIMD, après le décodage des instructions, plusieurs unités d'exécution accèdent simultanément la mémoire, obtiennent toutes les opérandes pour l'opération. Cette fonction permet SIMD particulièrement adapté pour les applications multimédia telles que les opérations de données intensives.

Tous les processeurs modernes contiennent des extensions à leur jeu d'instruction, comme le MMX, le SSE, etc. Ces extensions ont été ajoutées aux processeurs modernes pour pouvoir améliorer la vitesse de traitement sur les calculs. Les instructions SIMD sont composées notamment des jeux d'instructions :

- Sur processeur x86 : MMX, 3DNow!, SSE, SSE2, SSE3, SSSE3, SSE4 et AVX (en) ;
- Sur processeur PowerPC : AltiVec ;
- Sur processeur ARM : VFP, VFPv2, VFPv3lite, VFPv3, NEON, VFPv4 ;
- Sur processeur SPARC : VIS et VIS2 ;
- Sur processeur MIPS : MDMX et MIPS-3D.

2.2 Le principe du SIMD

SIMD peut copier plusieurs opérandes et les emballés dans un grand ensemble registre d'instructions. L'instruction SIMD va traiter chacune des données du vecteur indépendamment des autres. Par exemple, une instruction d'addition SIMD va additionner ensemble les données qui sont à la même place dans deux vecteurs, et placer le résultat dans un autre vecteur, à la même place. Quand on exécute une instruction sur un vecteur, les données présentes dans ce vecteur sont traitées simultanément. Donc, on peut exécute les opérations plus vite.

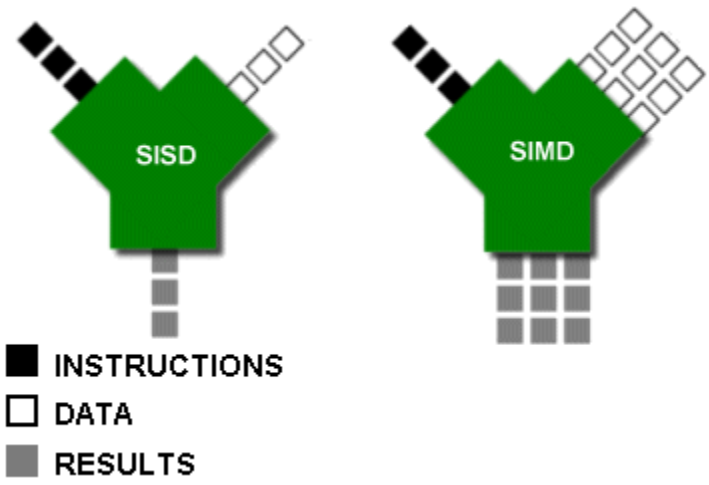

Figure2 Le principe du SIMD

Le principe de base de la technologie SIMD est dans une instruction, la CPU gère plusieurs éléments de données simultanément. De cette façon, nous pouvons améliorer l'efficacité de plusieurs fois.

2.3 Instructions SIMD

Il existe différents types d'instructions SIMD. Mais dans tous les cas, celles-ci travaillent sur un ensemble de données de même taille et de même type. Ces données sont rassemblées dans des espèces de blocs de données, d'une taille fixe, qu'on appelle un paquet. Ces paquets contiennent plusieurs nombres entiers ou nombres flottants placés les uns à côté des autres et regroupés dans ce vecteur.

Comme instructions SIMD, on trouve souvent [2] :

- des instructions arithmétiques du style additions, soustractions, multiplications, etc.
- des opérations logiques, comme des ET, des OU, des décalages ou des rotations.
- des comparaisons.
- ou des opérations de conversion.

Toutes ces instructions SIMD travaillent sur un ensemble de données de même taille et de même type. Ces données sont rassemblées dans des espèces de blocs de données, d'une taille fixe, qu'on appelle un vecteur. Ces vecteurs contiennent plusieurs nombres entiers ou nombres flottants placés les uns à côté des autres.

Une instruction SIMD va traiter chacune des données du vecteur indépendamment des autres. Par exemple, une instruction d'addition SIMD va additionner ensemble les données qui sont à la même place dans deux vecteurs, et placer le résultat dans un autre vecteur, à la même place. Quand on exécute une instruction sur un vecteur, les données présentes dans ce vecteur sont traitées simultanément.

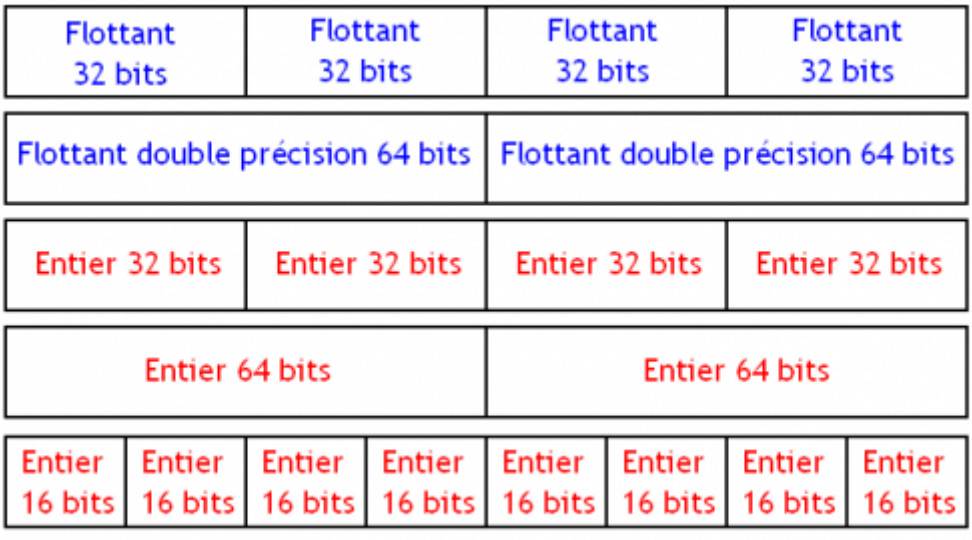

Figure3 la taille des paquets

Les instructions arithmétiques et quelques autres manipulent des nombres ou des données de taille fixe, qui ne peuvent prendre leurs valeurs que dans un intervalle déterminé par une valeur minimale et une valeur maximale. Si le résultat d'un calcul sort de cet intervalle, il ne pas être représenté dans notre ordinateur : il se produit ce qu'on appelle un over flow. Lorsqu'on travaille avec des paquets, il est évident que si un résultat prend plus de bits que ce qu'il est censé prendre, il vaut mieux éviter qu'il déborde sur la donnée d'à côté. Il faut donc trouver une solution pour éviter les catastrophes.

Première solution : on ne conserve que les bits de poids faibles du résultat et les bits en trop sont simplement oubliés. Par exemple, si on additionne deux nombres de 32 bits, le résultat est censé prendre 33 bits, mais on ne gardera que les 32 bit de base, et on oubliera le bit en trop. Seul problème : en faisant cela, une opération sur deux nombres très grands peut donner un nombre très petit, et cela peut poser des problèmes lorsqu'on manipule de la vidéo, des images ou du son. Imaginez que l'on représente une couleur par un entier de 8 bits (ce qui est souvent le cas) : par exemple, le blanc correspondra à la valeur maximale de cet entier, et le noir à 0. En ajoutant deux couleurs très proches du blanc (et donc deux entiers assez grands), le résultat obtenu sera un entier assez petit, qui sera donc proche du noir... Ce qui est fort peu crédible ! Et les mêmes problèmes peuvent arriver dans pleins d'autres situations : il vaut mieux qu'un son trop fort sature au lieu de le transformer en un son trop faible, par exemple.

Mais il existe une solution : utiliser ce qu'on appelle l'arithmétique saturée. Si un résultat est trop grand au point de générer un overflow, on arrondi le résultat au plus grand entier supporté par le processeur. Il va de soi que pas mal d'instructions SIMD utilisent cette arithmétique, vu qu'elles sont destinée au traitement d'image ou de son.

2.4 Application

Le modèle SIMD convient particulièrement bien aux traitements dont la structure est très régulière, comme c'est le cas pour le calcul matriciel. Généralement, les applications qui profitent des architectures SIMD sont celles qui utilisent beaucoup de tableaux, de matrices, ou de structures de données du même genre.1 On peut notamment citer les applications scientifiques, ou de traitement du signal. Dans les microprocesseurs, cette technologie est un contrôleur pour contrôler plusieurs microéléments parallèle, comme MMX ou SSE d'Intel et les instructions de « 3D Now! » d'AMD. Comme le traitement SIMD pour les données binaires à haut rendement, nous pouvons utiliser la technologie SIMD pour compléter beaucoup de travail, par exemple:

1. **Le traitement de l'image**

   Traitement de l'image numérique technologie SIMD est utilisé pour calculer le vecteur de vitesse et de la matrice, sa structure est dominée par le système de cache de données et du réseau de l'alignement composé de SIMD l'image convolution basée est une technologie clé. SIMD peut analyser l'architecture du système d'image de convolution et de la méthode d'image couleur de la reconnaissance de la vitesse, ce qui permet la technologie SIMD dans le traitement de l'image numérique.

2. **L'algorithme de chiffrement**

Avec le développement continu de technologies de l'information de sécurité, l'algorithme de chiffrement de l'information en utilisant une variété de techniques est également en hausse, le temps et la complexité de calcul est également en augmentation, l'algorithme de chiffrement dans le sens traditionnel a été incapable de répondre aux exigences de l'application, nous avons donc besoin d'utiliser instructions SIMD pour écrire algorithme de chiffrement plus efficace.

2.5 SSE

2.5.1 Le Streaming SIMD Extensions (SSE)

Dans les années 1999, une nouvelle extension SIMD fit son apparition sur les processeurs Intel Pentium 3 : le Streaming SIMD Extensions, abrévié SSE. Ce SSE fut ensuite complété, et différentes versions virent le jour : le SSE2, SSE3, SSE4, etc. [3]

Cette extension fit apparaitre 8 nouveaux registres, les registres XMM. Sur les processeurs 64 bits, ces registres sont doublés et on en trouve donc 16.

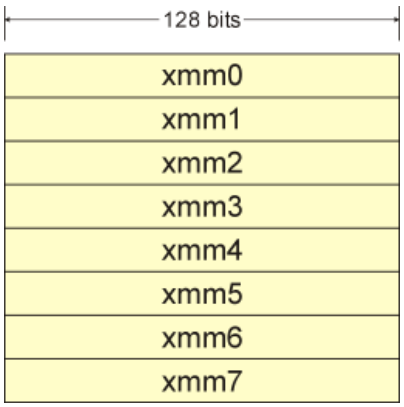

Figure4 les registres XMM des processeurs x86.

En plus de ces registres, on trouve aussi un nouveau registre supplémentaire qui permet de contrôler le comportement des instructions SSE : celui contient des bits qui permettront de dire au processeur que les instructions doivent arrondir leurs calculs d'une certaine façon, etc. Ce registre n'est autre que le registre MXCSR. Chose étrange, seuls les 16 premiers bits de ce registres ont une utilité : les concepteurs du SSE ont surement préférés laisser un peu de marge au cas où.

La première version du SSE contenait assez peu d'instructions : seulement 70. Croyez-moi, je ne vais pas toutes les lister, mais je peux quand-même dire qu'on trouve des instructions similaires au MMX (comparaisons, opérations arithmétiques, opérations logiques, décalages, rotations, etc.), avec pas mal d'opérations en plus. On y trouvait notamment des permutations, des opérations arithmétiques supplémentaires, et pas mal d'instructions pour charger des données depuis la mémoire dans un registre. Petit détail : la multiplication est gérée plus simplement, et l'on n'a pas besoin de s'embêter à faire mumuse avec plusieurs instructions différentes pour faire une simple multiplication comme avec le MMX.

Le SSE première version ne fournissait que des instructions pouvant manipuler des paquets contenant 4 nombres flottants de 32 bits (simple précision).

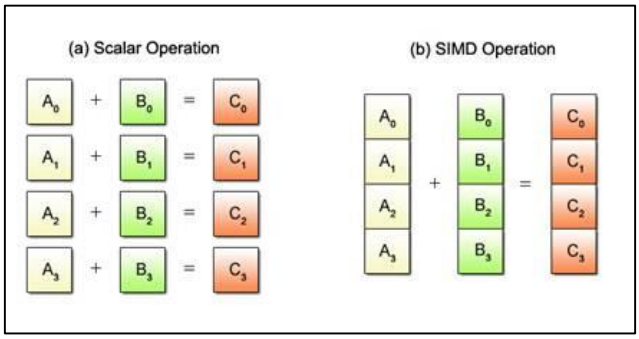

Figure5 l'exécution d'une instruction d'addition vectorielle.

On peut quand même signaler une chose : des instructions permettant de contrôler le cache firent leur apparition. On retrouve ainsi des instructions qui permettent d'écrire ou de lire le contenu d'un registre XMM en mémoire sans le copier dans le cache. Ces instructions permettent ainsi de garder le cache propre en évitant de copier inutilement des données dedans. On peut citer par exemple, les instructions MOVNTQ et MOVNTPS du SSE première version. On trouve aussi des instructions permettant de charger le contenu d'une portion de mémoire dans le cache, ce qui permet de contrôler son contenu. De telles instructions de prefetch permettent ainsi de charger à l'avance une donnée dont on aura besoin, permettant de supprimer pas mal de cache miss. Le SSE fournissait notamment les instructions PREFETCH0, PREFETCH1, PREFETCH2 et PREFETCHNTA. Autant vous dire qu'utiliser ces instructions peut donner lieu à de sacrés gains si on s'y prend correctement ! Il faut tout de même noter que le SSE n'est pas seul "jeu d'instruction" incorporant des instructions de contrôle du cache : certains jeux d'instruction POWER PC (je pense à l'Altivec) ont aussi cette particularité.

2.5.2 Instructions SSE

Le SSE ajoutait aussi pas mal d'instructions SIMD assez basiques. Ces instructions correspondaient aux instructions qu'on rencontre habituellement dans un jeu d'instruction normal. Ainsi, on trouve des instructions arithmétiques, qui travaillent sur des entiers regroupés dans un paquet occupant un registre SSE [4].

On trouve ainsi des instructions d'addition et de soustraction :

- PADDB, qui additionne deux paquets d'entiers de 8 bits ;
- PADDW, qui additionne deux paquets d'entiers de 16 bits ;
- PADDD, qui additionne deux paquets d'entiers de 32 bits ;
- PADDSB, qui prend un seul paquet et additionne tous les entiers de 8 bits contenus dans ce paquet ;
- PADDSW qui fait la même chose avec des paquets d'entiers de 16 bits.

On trouve aussi des soustractions :

- PSUBB, qui soustrait deux paquets d'entiers de 8 bits ;
- PSUBW, qui soustrait deux paquets d'entiers de 16 bits ;
- PSUBD, qui soustrait deux paquets d'entiers de 32 bits ;
- La multiplication est aussi supportée, mais avec quelques petites subtilités, via l'instruction PMULLW.

Ces opérations arithmétiques utilisent l'arithmétique saturée.

3 Mise en œuvre de SIMD: Projet SE

3.1 Opération bit a bit

En informatique, les opérations bit à bit apparaissent dans beaucoup des langages de programmation, et permettent de manipuler les données binaires, directement au niveau des bits. Elles sont utiles dès qu'il s'agit de manipuler les données à bas niveau : codages, couches basses du réseau (par exemple TCP/IP), cryptographie, où elles permettent également les opérations sur les corps finis de caractéristique 2.

Les opérations bit à bit courantes comprennent des opérations logiques bit par bit, et des opérations de décalage des bits, vers la droite ou vers la gauche.

Les opérateurs bits permettent de modifier et de tester un ou plusieurs bits d'une donnée. Ces opérateurs sont :

- NOT (NON)
- AND (ET)
- OR (OU)
- XOR (OU exclusif)
- SHR (Décalage à droite)
- SHL (Décalage à gauche)

Dans le projet SE, On pourra prendre comme cas d'usage la comparaison de « templates » (i.e. images) binaires pour la protection de copyright de Manga. Dans ce contexte, les instructions mises en œuvre seront principalement de type « bitwise operator ». Donc, il faut utiliser l'opérateur binaire XOR, l'opérateur binaire XOR combine l'état de 2 bits selon le tableau suivant :

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Tableau1 la vérité A XOR B

Apres la comparaison, on va obtenir le résultat. Si le résultat égale 0, on peut dire que les deux images est le même. De cette façon, nous pouvons prévenir l'émergence de la piraterie. Ici, je vais vous présenter que comment appliquer notre procédure pour protéger de copyright de Manga à l'opération bit à bit.

3.2 La comparaison binaire par langage C

L'opérateur binaire XOR de langage C est « ^ ». Il agit sur chaque bit de la valeur :

```
unsigned a = 0xF0F0;
unsigned b = 0x00FF;
unsigned c = a ^ b; /* c == 1111 0000 0000 1111 soit 0xF00F */
```

Tableau2 l'opérateur binaire XOR de langage C

Si le type de variable a et variable b est float, on peut utiliser L'instruction « if…else… » :

```
float a [4] = {0, 0, 0, 0} ;
float b [4] = {1, 0, 0, 0} ;
float c [4] ;
for (int i=0; i<4; i++) {
    if (a[i] == b[i])
        c[i]=0 ;
    else
        c[i]=1 ;
}
```

Tableau3 l'opérateur binaire XOR de SSE

Depuis notre projet implique un grand nombre de virgules flottantes données arithmétiques, nous utilisons donc la seconde méthode. Voici notre code de projet :

Dans la fonction, l'entrée est trois tableaux qui mémorisent les données, on va comparer le tableau a et le tableau b, et puis, on sauve les résultats dans tableau c. L'entier num est les numéros dans le tableau a et le tableau b.

```
int CompareData (float *a, float *b, float *c, int num) {
    int i;
    for (i = 0; i < num; i++) {
        if (a[i] == b[i])       c[i] = 0;
        else       c[i] = 1;
    }
    return 0;
}
```

Tableau4 La fonction «CompareDate»

3.3 La comparaison binaire par SIMD

Dans notre projet, nous avons également utilisé la technologie SIMD. Nous allons faire la même chose avec le langage C, qui compare les données binaires. Pour les technologies SIMD, il existe de nombreuses versions, dont nous avons déjà mis en place l'avant. Dans ce projet, nous allons utiliser la technologie SSE.

Dans l'instruction SSE, nous allons utiliser l'opérateur binaire XOR. La définition de l'opérateur binaire XOR dans SSE est la suivante:

__m128 _mm_xor_ps (__m128 a, __m128 b);

Cette fonction peut calculs EXOR de bits (exclusivité-ou) des quatre simples précisions, valeurs à virgule flottante d' a et b. La variable d'entrée est les données qui sont __m128.

__m128 est un type de variable, qui utilisable avec les intrinsèques d'instructions Extensions Streaming SIMD(SSE) et Extensions Streaming SIMD 2(SSE2), est défini dans « xmmintrin.h ». Il est à noter:

1. Vous ne devez pas accéder aux champs __m128 directement.
2. Vous pouvez toutefois visualiser ces types dans le débogueur.
3. Une variable de type __m128 mappe aux registres XMM [0-7].
4. Les variables de type _m128 sont alignées automatiquement sur des limites de 16 octets.
5. Le type de données __m128 n'est pas pris en charge sur les processeurs ARM.

La fonction «__m128 _mm_load_ps (float *p) » est pour loads four single-précision, floating-point values. Donc, on utilise cette fonction pour chaque fois que nous obtenons quatre données

Dans la fonction, l'entrée est trois tableaux qui mémorisent les données, on va comparer le tableau a et le tableau b, et puis, on sauve les résultats dans tableau c. L'entier num est les numéros dans le tableau a et le tableau b.

```
int CompareData_SSE (float *a, float *b, float *c, int num) {
    int i;
    _MM_ALIGN16 float *pA = a;
    _MM_ALIGN16 float *pB = b;
    _MM_ALIGN16 float *pC = c;
    __m128 x, y, z ;
    for (i = 0; i < num / 4; i++) {
        x = _mm_load_ps (pA);
        y = _mm_load_ps (pB);
        z = _mm_xor_ps (x, y);
        _mm_store_ps (pC, z);
        pA += 4;
        pB += 4;
        pC += 4;
```

```
    }
    return 0;
}
```

Tableau5 La fonction « CompareDate_SSE»

Comme la suite est la fonction principale :

```
int main (){
    int i, num=0;
    double **d=NULL;
    d= new double *[2];
    d [0] = new double [9];
    d [1] = new double [9];
    LARGE_INTEGER t1, t2, t3, feq;
    _MM_ALIGN16 float *a = (_MM_ALIGN16 float*)malloc(sizeof(_MM_ALIGN16 float)*(NUM8 + 1));
    _MM_ALIGN16 float *b = (_MM_ALIGN16 float*)malloc(sizeof(_MM_ALIGN16 float)*(NUM8 + 1));
    _MM_ALIGN16 float *c = (_MM_ALIGN16 float*)malloc(sizeof(_MM_ALIGN16 float)*(NUM8 + 1));
    for (i = 0; i < NUM8; i++){
        a[i] = i % 2;
        b[i] = (i * 1) % 2;
    }
    a[3] = 1; a[4] = 1;
    a[NUM8] = '\0';
    b[NUM8] = '\0';
    c[NUM8] = '\0';
    cout << "A:   ";
    for (i = 0; i < 24; i++){
        cout << a[i] << " ";
    }cout << "..." << endl << "B:   ";
    for (i = 0; i < 24; i++){
        cout << b[i] << " ";
    }cout << "..." << endl;
    QueryPerformanceFrequency(&feq);
    for (i = 0; i < 9; i++){
        num = getNum(i);
        QueryPerformanceCounter(&t1);
        CompareData(a, b, c, num);
        QueryPerformanceCounter(&t2);
        CompareData_SSE(a, b, c, num);
        QueryPerformanceCounter(&t3);
        d[0][i] = 1000 * ((double)t2.QuadPart - (double)t1.QuadPart) / ((double)feq.QuadPart);      d[1][i] = 1000 * ((double)t3.QuadPart - (double)t2.QuadPart) /
```

```
((double)feq.QuadPart); }
	cout<< "C:    ";
	for (i = 0; i < 24; i++){
		cout << c[i] << " ";
	}cout << "..." << endl;
	ShowTable(d);
	system("pause");
	return 0;
}
```

Tableau6 La fonction «main»

3.4 La distinction entre SIMD et SISD

Pour distinguer entre les deux efficacités, nous avons sélectionné neuf ensembles de données, et pour chaque type de données, nous avons utilisées deux méthodes pour ces essais. Lors de nos tests, nous avons utilisé la méthode de la tranche de temps processeur pour enregistrer le code à exécuter temps. Finalement, on a obtenu le résultat comme la suite :

```
A:  0 1 0 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 ...
B:  0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 ...
C:  0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...

                  No SSE          |       SSE
NUM_1   2^16 |   1.0875ms        |       0.216858ms      5.01479
NUM_2   2^17 |   2.07363ms       |       0.484402ms      4.28079
NUM_3   2^18 |   4.0292ms        |       0.939291ms      4.28962
NUM_4   2^19 |   8.14886ms       |       2.01332ms       4.04748
NUM_5   2^20 |   16.1444ms       |       4.06513ms       3.97143
NUM_6   2^21 |   32.598ms        |       7.59067ms       4.29448
NUM_7   2^22 |   55.1275ms       |       14.8285ms       3.71768
NUM_8   2^23 |   136.422ms       |       30.8625ms       4.42033
NUM_9   2^24 |   241.348ms       |       64.7469ms       3.72757
```

Figure6 Les résultats d'exécution du projet.

Nous pouvons voir sur la figure : Sur l'efficacité opérationnelle, le programme qui utilise SIMD est plus vite que SISD. Théoriquement, l'efficacité opérationnelle devrait être quatre fois le SIMD de SISD, mais les données sont stockées sur le disque dur, Lorsque nous consommons beaucoup de temps pour lire les données. Donc, le temps qu'on a obtenu ne sont pas vraiment La durée du programme de fonctionner.

En bref, l'utilisation de la technologie SIMD peut vraiment améliorer l'efficacité du code, en particulier dans le traitement d'image. Dans le traitement de l'image, puisque nous avons affaire à un grand nombre d'opérations de données binaires. Grâce à la technologie SIMD peut grandement améliorer la vitesse de traitement.

4 conclusions

Dans ce projet, on a bien compris l'idée principale et les instructions de SIMD. On utilise les instructions d'opération bit à bit pour réaliser l'opération de XOR avec une grande quantité de données. Pendant développement du projet, on lit beaucoup de références. Sur les références, on a appris les instructions de SIMD, et son application. On a appris également plus sur le micro-processeur et le registres en même temps.

L'objectif de ce projet est la mise en œuvre de ces instructions au sein d'une architecture parallèle multi-cœur, on prend les instructions de SIMD pour la comparaison d'une grande quantité de donnes. L'utilisation de SIMD peut améliorer l'efficacité du programme, en particulier pour un grand nombre de données, l'effet est plus évident. Dans ce projet, on utilise 2 méthodes, la méthode traditionnel avec langage C++ et la méthode avec les instructions de SIMD, pour comparer le même données, et on compte les temps d'exécution de les 2 méthodes respectivement. Selon la comparaison des temps d'exécution des 2 méthodes, on peut voir que sur la même donnée, la vitesse de traitement d'utilisation de l'instruction SIMD est quatre fois plus que lequel de la méthode ordinaire avec langage C++.

Peut-être à cause de le matériel d'ordinateur, chaque fois que le temps d'exécution du programme est un peu diffèrent. Mais par les résultats en exécutant plusieurs fois, on peut encore découvrir que l'instruction de SIMD à un grand avantage sur la manipulation de grandes quantités de données.

Bibliographie

[1] Wikipedia-- Single instruction multiple data
http://fr.wikipedia.org/wiki/Single_instruction_multiple_data

[2] Jun Zhou. 2013. Use of SIMD Vector Operations to Accelerate Application Code Performance on Low-Powered ARM and Intel Platforms [D]. L'université de Griffith, Australia.

[3] http://www.bien-programmer.fr/bits.htm

[4] https://software.intel.com/sites/landingpage/IntrinsicsGuide/

Résume

Il s’agit d’un projet ayant pour but de mise en œuvre de ces instructions au sein d’une architecture parallèle SIMD / multi-cœur pour la comparaison binaires (opération XOR), ce qui a l’application à la protection de copyright de Manga.
Mots clés : SIMD XOR

Abstract

It is a project aimed at implementing these instructions in a parallel SIMD / multi-heart for the binary comparison (XOR), which has application to copyright protection Manga.
Keywords: SIMD XOR