Unités vectorielles

Oguz Kaya

Polytech Paris-Sud

Orsay, France

Programmation parallèle : 4ème année apprentissage

Outline

Outline

Outline

Calcul Scalaire

Principe

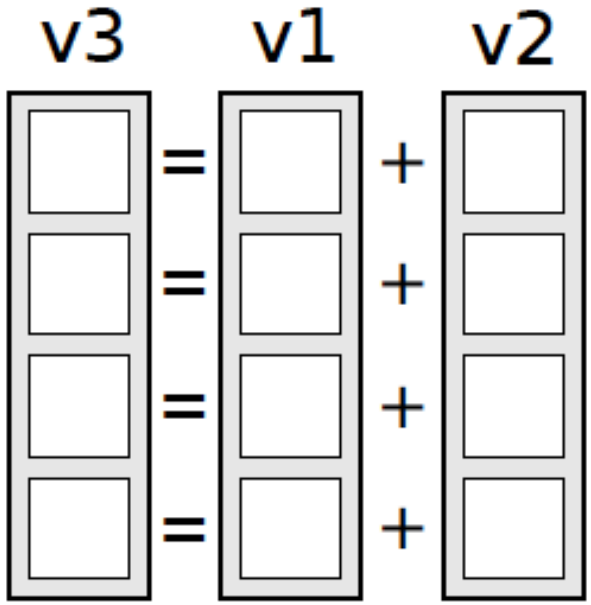
Un processeur scalaire effectue les opérations séquentiellement, chaque opération portant sur des données scalaires (un scalaire)

Exemple : addition de vecteurs de 4 éléments

```
v3[0] = v1[0] + v2[0];
v3[1] = v1[1] + v2[1];
v3[2] = v1[2] + v2[2];
v3[3] = v1[3] + v2[3];
```

UNIVERSITÉ PARIS SUD

Calcul Scalaire



Calcul Vectoriel

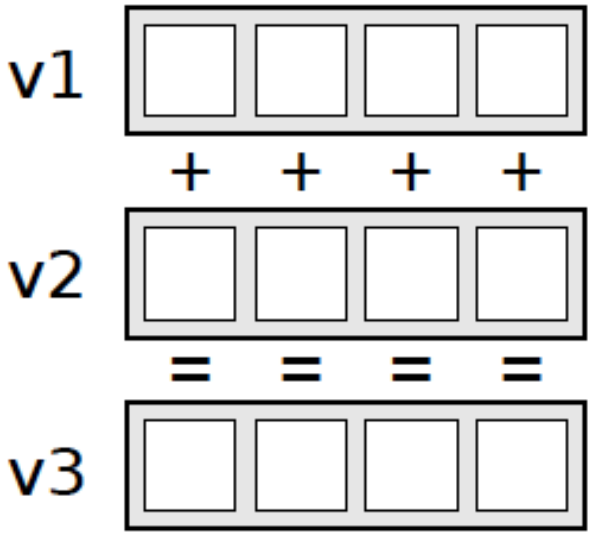
Principe

Un processeur vectoriel effectue les opérations en parallèle sur un ensemble de scalaires, chaque opération portant sur des vecteurs

Exemple : addition de vecteurs de 4 éléments

v3 = v1 + v2;

Calcul Vectoriel



Avantages/inconvénients de l'approche vectorielle

1. Avantage :
   - Instructions n fois plus rapide, n la largeur de l'unité vectorielle si l'algorithme se prête au paradigme SIMD : une même instruction simultanément sur plusieurs données
     - calcul vectoriel
     - simulations scientifiques
     - traitement d'images
     - apprentissage automatique (réseaux neurons)
2. Inconvénient :
   - Gain uniquement si l'algorithme se prête au paradigme SIMD,
   - Parfois difficile (et sale) à utiliser à la main (automatisations possible),
   - sinon on se retrouve dans le cas scalaire, perte d'efficacité

UNIVERSITÉ PARIS SUD

Premières implantations

- Années 60 : premier projet Solomon
- Années 70 : ILLIAC IV, université de l'Illinois
- ensuite : CDC, Cray, NEC, Hitachi, Fujitsu
- depuis 1996 : processeurs "grand public" scalaires + unités vectorielles, Pentium (MMX, SSE, AVX), PowerPC (AltiVec)

MultiMedia eXtension : MMX

Première unité vectorielle “grand public” introduite par Intel sur certains processeurs Pentium de première génération

- 8 registres 64bits
- 57 opérations sur les entiers
- Limitations :
  - Uniquement sur les entiers
  - Registres partagés avec l’unité FPU (donc pas possible d’utiliser les operations flottants scalaires simultanement)

Fonctionnement général des unitées SIMD

1. Chargement des données de la mémoire vers les registres SIMD
   - Faire attention à rester dans le cache pour de bonnes performances.
2. Opération sur les registres SIMD
3. Placement du résultat en mémoire
   - Idem, falloir rester dans le cache.

Programmation

- Vectorisation automatique : le compilateur tente de vectoriser le code automatiquement
- Assembleur : programmation bas-niveau, manipulation directe des registres et instructions
- Intrinsics : fichiers .h inclus dans GCC : ?mmintrin.h
- Bibliothèques de haut-niveau (OpenMP) : masquage de la programmation vectorielle

UNIVERSITÉ PARIS SUD

Performance

Astuces pour obtenir les meilleurs performances

- Organiser correctement les données pour éviter les accès non contigus et également optimiser le cache
- Aligner les données (moins importants dans les versions récentes)
- Rester le plus longtemps possible dans les registres (c'est à dire, éviter les lectures/écritures sur la mémoire)

Organisation des données

- Tableau de structures (Array of structures - AoS) : xyzwxyzwxyzw ...
- Structure de tableaux (Structure of Arrays - SoA) xxxxxxxxx ... yyyyyyyyy ... zzzzzzzzz ... wwwwwwwww ...
- Structure hybride : xxxxyyyyzzzzwwwwxxxxyyyyzzzzwwww ...

Outline

Streaming SIMD Extensions

- SSE - Pentium III, Athlon XP
- SSE2 - Pentium 4
- SSE3 - Pentium 4 Prescott
- SSSE3 - (Supplemental SSE3)
- SSE4.1 - Core2
- SSE4.2 - Core i7
- AVX
- **AVX2**
- AVX512

Composition

Chaque nouvelle version de SSE ajoute des instructions à la précédente
SSE(N) = nouvelles instructions + SSE(N-1)

Advanced Vector Extensions (AVX)

Registres de 256 bits

16 registres 256bits sur x86-64: YMM0 $->$ YMM15

Instructions

- transferts mémoire $<-$ registres AVX
- opérations arithmétiques
- opérations logiques
- tests
- permutations

UNIVERSITÉ PARIS SUD

Intrinsics

Les intrinsics sont accessibles à travers des fichiers .h qui déffinissent :

- des types de données
- des fonctions opérant sur ces types de données

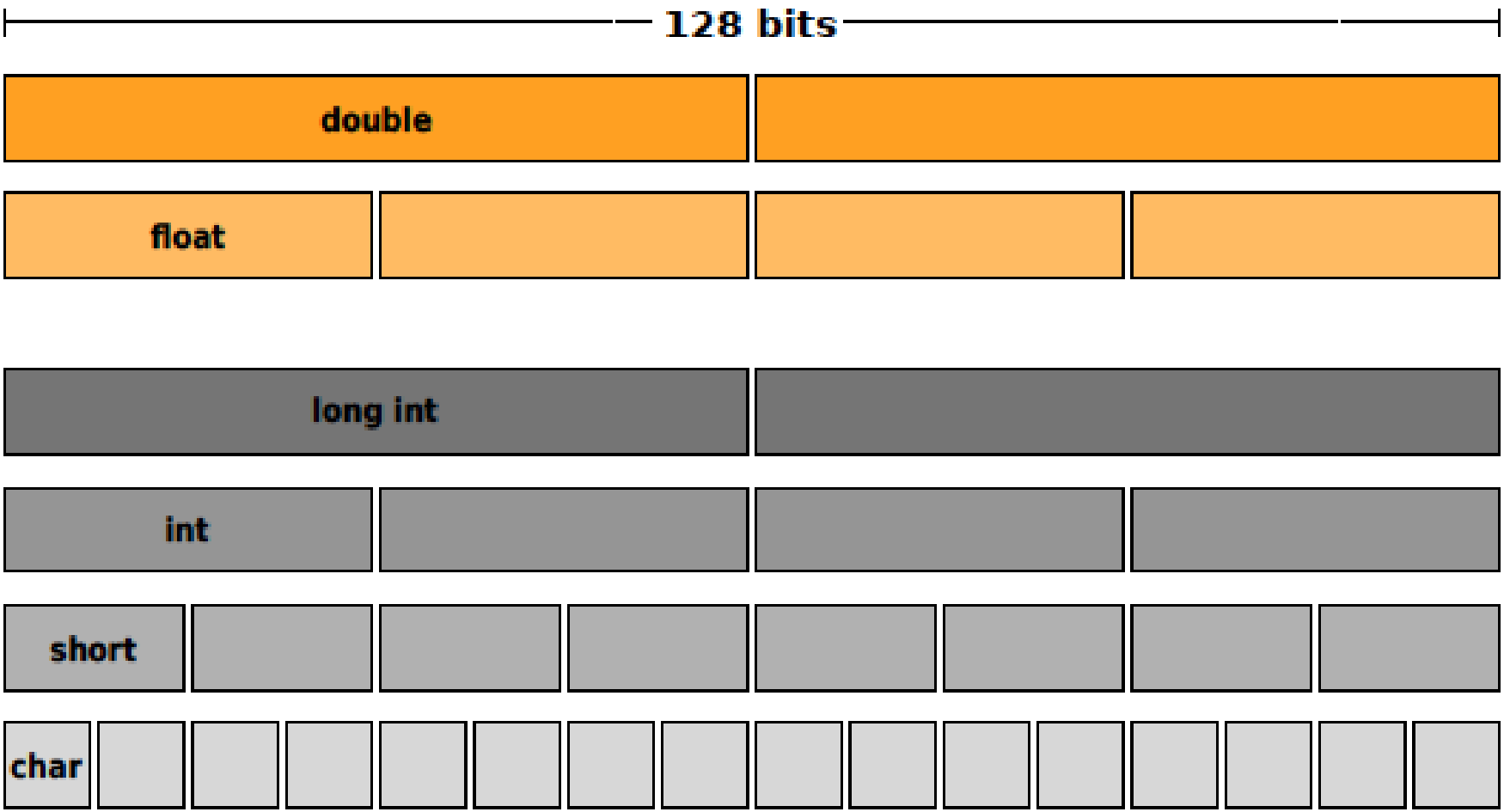
Fichiers

- SSE : xmmintrin.h
- SSE2 : emmintrin.h
- SSE3 : pmmintrin.h
- SSSE3 : tmmintrin.h
- SSE4.1 : smmintrin.h
- SSE4.2 : nmmintrin.h
- **AVX/AVX2 : immintrin.h**

Types de données

Types

- __m256 : 8 foats
- __m256d : 4 doubles
- __m256i : $256/\mathit{largeur}$ entiers ($\mathit{largeur} = 8, 16, 32, 64$)


128 bits
double
float
long int
int
short
char

Fonctions

Nomenclature générale

_mm256{operation}{alignement}_{dataorganization}{datatype}...

Exemple : addition de 2 vecteurs de 8 flottants

__m256 C = _mm256_add_ps(__m256 A, __m256 B)

Terminologie

- s (single) : flottant simple précision (32bits)
- d (double) : flottant double précision (64bits)
- i... (integer) : entier
- p (packed) : contigus, opérer sur le vecteur entier
- s (scalar) : n'opérer que sur un élément
- u (unaligned) : données non alignées en mémoire
- l (low) : bits de poids faible
- h (high) : bits de poids fort
- r (reversed) : dans l'ordre inverse

Fonctionnement général

1. déclaration des variables AVX (registres):
   __m256 r1;
2. chargement des données de la mémoire vers les registres AVX:
   r1 = _mm256_load_...(type* p)
3. opérations sur les registres SIMD
4. placement du contenu des registres en mémoire :
   _mm256_store_...(type* p, r1)

Transferts mémoire $<-->$ registres SSE

- alignés ou non alignés :
  _mm256_load... ou _mm256_loadu...
  _mm256_store... ou _mm256_storeu...

- par vecteurs : $8 \times SP$, $4 \times DP$, ...
  _mm256_load_ps, _mm256_loadu_pd, ...
  _mm256_store_ps, _mm256_storeu_pd, ...

- par élément scalaire :
  _mm256_load_ss, _mm256_load_sd, ...
  _mm256_store_ss, _mm256_store_sd, ...

UNIVERSITÉ PARIS SUD

Accès alignés

Le processeur peut effectuer des transferts efficaces de 16 octets (128 bits) entre la mémoire et un registre SSE sous la condition que le bloc soit aligné sur 16 octets. Cette contrainte est matérielle

Attention

L'alignement dépend de l'architecture et du type de variable. Par défaut l'alignement d'un entier 32 bits est effectué sur 4 octets
Pour obtenir l'alignement : __alignof__(type)

Accès alignés

Cacheline splits

Une ligne de cache a généralement une taille de 32 à 64 octets.

Si la donnée chargée dans un registre SSE provient d'une zone mémoire à "cheval" sur 2 lignes de cache, alors il faut lire les 2 lignes de cache pour pouvoir remplir le registre SSE, ce qui implique une baisse de performance

Allocation de données alignées

- Alignement de données statiques sur 16 octets :
  `int x __attribute__ ((aligned (16)))`

- Alignement de données dynamiques sur 16 octets :
  `type* x = (type*) _mm_malloc(size*sizeof(type), 16);`

SSE fournit des opérations d'accès sur des données alignées et non alignées. Cependant les accès non alignés sont beaucoup plus lents!

Copie de floats vectorisée

- Allouer deux tableaux (*tab*0, *tab*1, de taille *dim*) de manière alignée par 32 octets
- Initialiser les tableaux ($tab0[i] = i$, $tab1[i] = 0$) (non-vectorisée)
- Copier *tab*0 dans *tab*1 de manière vectorisée.
- Comparer le temps d'execution
- Calculer et afficher l'acceleration et l'efficacité
- Compiler le code en rajoutant le drapeau $-O3$ puis comparer les résultats. Sont-ils pareils?

Compilation: g++ fichier.cpp -o fichier -mavx

UNIVERSITÉ PARIS SUD

Opérations arithmétiques

- _mm256_add_pd(A, B) - (Add-Packed-Double)
  - Entrée : [A0, A1, A2, A3], [B0, B1, B2, B3]
  - Sortie : [A0 + B0, A1 + B1, A2 + B2, A3 + B3]
- _mm256_add_ps(A, B) - (Add-Packed-Single)
  - Entrée : [A0, . . . , A7], [B0, . . . , B7]
  - Sortie : [A0 + B0, . . . , A7 + B7]
- _mm256_add_epi64(A, B) - (Add-Packed-Int64)
  - Entrée : [A0, A1, A2, A3], [B0, B1, B2, B3]
  - Sortie : [A0 + B0, A1 + B1, A2 + B2, A3 + B3]
- _mm256_add_epi32(A, B) - (Add-Packed-Int32)
- _mm256_add_epi16(A, B) - (Add-Packed-Int16)
- _mm256_add_epi8(A, B) - (Add-Packed-Int8)

Idem pour _mm256_sub_. . .

UNIVERSITÉ PARIS SUD

Opérations arithmétiques

- _mm256_mul_pd - (Multiply-Packed-Double)
  - Entrée : [A0, A1, A2, A3], [B0, B1, B2, B3]
  - Sortie : [A0 * B0, A1 * B1, A2 * B2, A3 * B3]
- _mm256_mul_ps - (Multiply-Packed-Single)
  - Entrée : [A0, . . . , A7], [B0, . . . , B7]
  - Sortie : [A0 * B0, . . . , A7 * B7]
- _mm256_mul_epi64 - (Multiply-Packed-Int64)
  - Entrée : [A0, A1, A2, A3], [B0, B1, B2, B3]
  - Sortie : [A0 * B0, A1 * B1, A2 * B2, A3 * B3]
- _mm256_mul_epi32 - (Multiply-Packed-Int32)
- _mm256_mul_epi16 - (Multiply-Packed-Int16)
- _mm256_mul_epi8 - (Multiply-Packed-Int8)

Idem pour _mm256_div_. . .

UNIVERSITÉ PARIS SUD

Opérations arithmétiques

- _mm256_fmadd_pd - (Fused-Multiply-Add-Packed-Double)
  - Entrée : [A0, A1, A2, A3], [B0, B1, B2, B3], [C0, C1, C2, C3]
  - Sortie : [A0 * B0 + C0, A1 * B1 + C1, A2 * B2 + C2, A3 * B3 + C3]
- _mm256_fmadd_ps - (Fused-Multiply-Add-Packed-Single)
  - Entrée : [A0, . . . , A7], [B0, . . . , B7], [C0, . . . , C7]
  - Sortie : [A0 * B0 + C0, . . . , A7 * B7 + C7]

Idem pour _mm256_fmsub_. . .

UNIVERSITÉ PARIS SUD

Opérations horizontales (disponibles à partir de SSE3)

- _mm256_hadd_pd(A, B) - (Horizontal-Add-Packed-Double)
  - Entrée : [A0, A1, A2, A3], [B0, B1, B2, B3]
  - Sortie : [A0 + A1, B0 + B1, A2 + A3, B2 + B3]
- _mm256_hadd_ps(A, B) (Horizontal-Add-Packed-Single)
  - Entrée : [A0, . . . , A7], [B0, . . . , B7]
  - Sortie : [A0 + A1, A2 + A3, B0 + B1, B2 + B3, A2 + A3, . . . , B6 + B7]
- Idem pour _mm256_hsub_. . .

UNIVERSITÉ PARIS SUD

Opérations logiques

- _mm256_{and , or , xor, . . . }_{ps, pd, si128}(A, B)
- _mm256_cmp_ps, pd(A, B, op)
  - op = _CMP_LT_OS: Comparaison A $<$ B
  - op = _CMP_LE_OS: Comparaison A $\leqslant$ B
  - op = _CMP_EQ_OS: Comparaison A $==$ B
  - op = _CMP_GT_OS: Comparaison A $>$ B
  - op = _CMP_GE_OS: Comparaison A $\geqslant$ B
- _mm256_cmpeq, gt_epi_8, 16, 32, 64(A, B)

Comparaisons

La comparaison des registres AVX à l'aide des instructions _mm256 _cmp. . . produit un masque contenant un champs de 1 lorsque la condition est vérifiée, et un champs de 0 dans le cas contraire

UNIVERSITÉ PARIS SUD

Mélange des données : Shuffle

L'instruction shuffle permet de combiner des données de 2 registres donnés en argument suivant un masque indiquant les positions à réecupérer
Il est possible de passer le même registre en argument

Limitation

L'instruction shuffle impose de réecupéerer autant de données dans les 2 registres. Par exemple, il est possible de récupérer 2 flottants dans chacun des 2 registres mais pas 1 flottant de l'un et 3 fottants de l'autre

UNIVERSITÉ PARIS SUD

Exemple

```
float a0[4] __attribute__((aligned(16))) = {1, 2, 3, 4};
float a1[4] __attribute__((aligned(16))) = {5, 6, 7, 8};
float a2[4] __attribute__((aligned(16)));

__m128 r0 = _mm_load_ps(a0);
__m128 r1 = _mm_load_ps(a1);

// _MM_SHUFFLE is a macro used to create the mask.
// In this example, it takes values 0 and 1 from r1
//                          and values 2 and 3 from r0
r0 = _mm_shuffle_ps(r0, r1, _MM_SHUFFLE(3, 2, 1, 0));

_mm_store_ps(a2, r0); // contains {1, 2, 7, 8}
```

Affectation des constantes

- _mm256_set_ps(float, . . . , float)
- _mm256_set_pd(double, double, double, double)
- _mm256_set_epi64(int, int, int, int)
- _mm256_set_epi32(int, . . . , int)
- _mm256_set_epi16(short, . . . , short)

Exercice

Soit une machine fournissant les instructions vectorielles suivantes :

- vector float **vec_ld**(float*) : chargement aligné de 4 floats
- void **vec_st**(vector float, float*) : rangement aligné de 4 floats
- vector float **vec_splat**(float) : remplissage d'un vecteur par une constante
- vector float **vec_add**(vector float, vector float) : somme élément par élément de deux vecteurs de floats
- vector float **vec_mul**(vector float, vector float) : produit élément par élément de deux vecteurs de floats
- float **vec_hadd**(vector float) : somme des éléments d'un vecteur de floats

UNIVERSITÉ PARIS SUD

Exercice

- Toutes ces instructions ont une latence de 1 cycle.
- On considère que vector float est le type représentant un registre vectoriel 128 bits contenant 4 floats contigus. Leur initialisation peut s'écrire : vector float f = 1,2,3,4;
- Implémenter un code vectoriel utilisant ces instructions permettant de calculer le produit scalaire de deux tableaux de float de $4N$ éléments.

UNIVERSITÉ PARIS SUD