Programmation vectorielle/SIMD

Sylvain Jubertie
sylvain.jubertie@univ-orleans.fr

Plan

Plan

Plan

Calcul scalaire

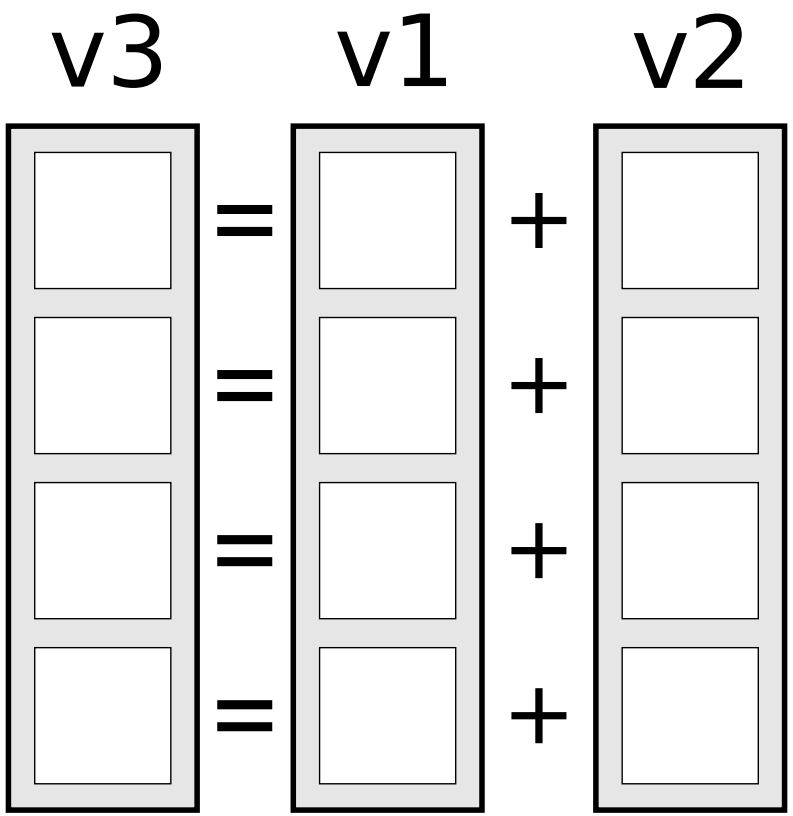
Principe

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];
```

Calcul scalaire



Calcul vectoriel

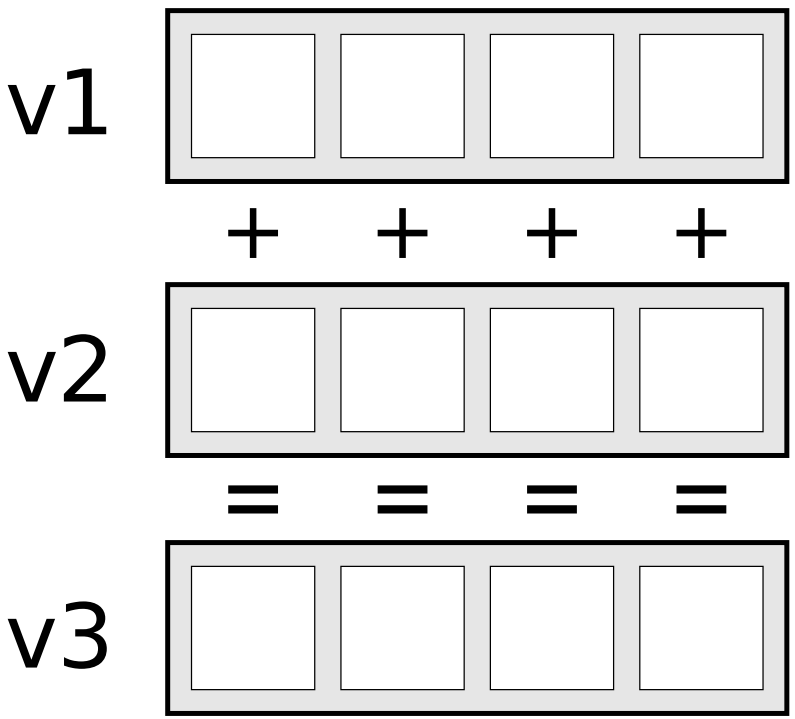
Principe

Un processeur vectoriel applique une même instruction simultanément sur plusieurs données (un vecteur de scalaires).

Exemple : addition de vecteurs de 4 éléments

```
v3 = v1 + v2;
```

Calcul vectoriel



Avantages/inconvénients de l'approche vectorielle

Avantage

Calcul *n* fois plus rapide, n largeur de l'unité vectorielle si l'algorithme se prête au paradigme SIMD : une même instruction simultanément sur plusieurs données.

- calcul vectoriel
- traitement d'images

Inconvénient

Gain uniquement si l'algorithme se prête au paradigme SIMD. Sinon on se retrouve dans le cas scalaire, perte d'efficacité.

Plan

Premières implantations

Historique

- 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)

MMX

MultiMédia eXtension ?

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

MMX

Description

- 8 registres 64bits
- 57 opérations sur les entiers

Limitations

- uniquement sur les entiers
- registres partagés avec l'unité FPU

Autres jeux d'instructions SIMD

Unités SIMD

- AMD 3DNow ! : AMD K6-2, . . ., Phenom II, certains VIA C5, Transmeta Crusoe et Efficeon
- ARM Neon : Cortex-A8&9 (Archos 5, Pandora)
- approches FPGA
- GPU

Plan

Programmation

Fonctionnement général des unités SIMD

1. chargement des données de la mémoire vers les registres SIMD
2. opération sur les registres SIMD
3. placement du résultat en mémoire

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 : `pmmintrin.h` pour SSE3

Bibliothèques de haut-niveau

masquage de la programmation vectorielle

Plan

Performance

Tips pour obtenir les meilleurs performances

- organiser correctement les données pour éviter les accès non contigus et également optimiser le cache
- aligner les données
- rester le plus longtemps possible dans les registres !

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...

Plan

SSE & Intrinsics

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

SSE & Intrinsics

Connaître la version de SSE de son processeur

`cat /proc/cpuinfo`

- sse
- sse2
- pni (Prescott New Instructions - SSE3)
- ssse3
- sse4_1

SSE & Intrinsics

Composition

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

SSE & Intrinsics

Registres de 128 bits

- 8 registres 128bits sur x86 : XMM0 -> XMM7
- 16 registres 128bits sur x86_64 : XMM8 -> XMM15

Instructions

- transferts mémoire <->registres SSE
- opérations arithmétiques
- opérations logiques
- tests
- permutations

SSE & Intrinsics

Définitions

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

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

SSE & Intrinsics

Fichiers

- SSE : xmmintrin.h
- SSE2 : emmintrin.h
- SSE3 : pmmintrin.h
- SSSE3 : tmmintrin.h
- SSE4.1 : smmintrin.h

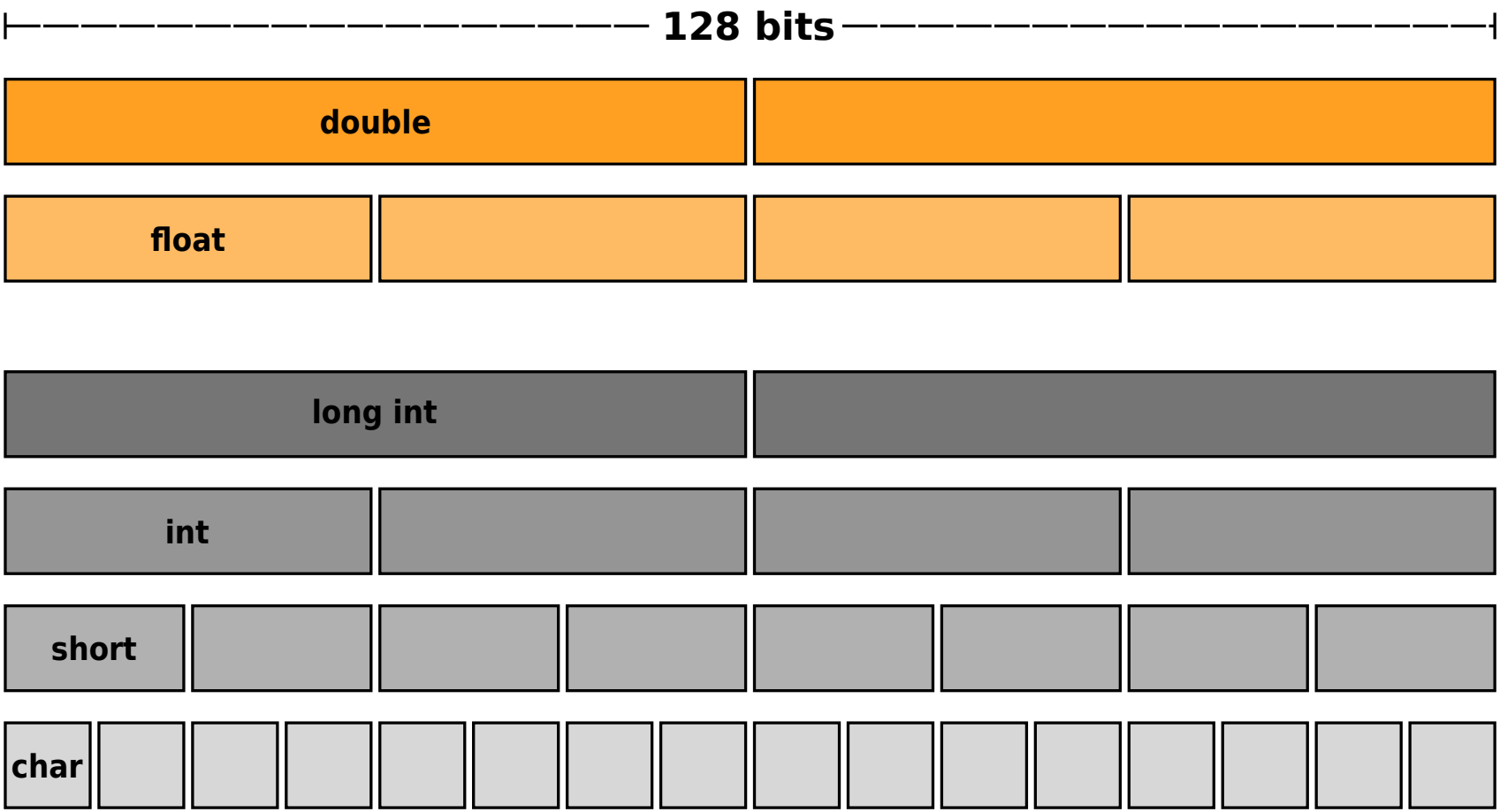
Plan

SSE & Intrinsics : Types de données

Types

- __m128 : 4 floats
- __m128d : 2 doubles
- __m128i : entiers

SSE & Intrinsics : Types de données



Plan

SSE & Intrinsics : Fonctions

Nomenclature générale

_mm_{operation}{alignement}_{dataorganization}{datatype}(...)

Exemple : addition de 2 vecteurs de 4 floats

_mm_add_ps(__m128 A, __m128 B)

SSE & Intrinsics : Fonctions

Terminologie

- `s` (single) : flottant simple précision (32bits)
- `d` (double) : flottant double précision (64bits)
- `i...` (integer) : entier
- `p` (packed) : contigus
- `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

SSE & Intrinsics : Fonctions : Opérations mémoire

Fonctionnement général

1. déclaration des variables SSE (registres)
   `__m128 r1`
2. chargement des données de la mémoire vers les registres SIMD
   `r1 = _mm_load...(type* p)`
3. opérations sur les registres SIMD
4. placement du contenu des registres en mémoire
   `_mm_store...(type* p, r1)`

SSE & Intrinsics : Fonctions : Opérations mémoire

Transferts mémoire <->registres SSE

- alignés ou non alignés
  `_mm_load_...` ou `_mm_loadu_`
  `_mm_store_...` ou `_mm_storeu_`
- par vecteurs : 4xSP, 2xDP, ... `_mm_load{u}_ps`, `_mm_load{u}_pd`, ...
  `_mm_store{u}_ps`, `_mm_store{u}_pd`, ...
- par élément scalaire
  `_mm_load_ss`, `_mm_load_sd`, ...
  `_mm_store_ss`, `_mm_store_sd`, ...

SSE & Intrinsics : Fonctions : Opérations mémoire

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)`
`__alignof__(int[4])` $\rightarrow$ `4`

SSE & Intrinsics : Fonctions : Opérations mémoire

Accès aligné

Alignement de données statiques sur 16 octets :

```
int x __attribute__ ((aligned (16)))
```

Alignement de données dynamiques sur 16 octets :

```
int posix_memalign(void **memptr, 16, sizeof(type))
```

SSE & Intrinsics : Fonctions : Opérations mémoire

Accès non aligné

Nécessite plusieurs accès mémoire car les données sont réparties sur 2 blocs de 16 octets.

SSE & Intrinsics : Fonctions : Opérations mémoire

Impact sur les performances

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

SSE & Intrinsics : Fonctions : Opérations mémoire

Exemple : Copie de 4 floats

```
#include <stdio.h>
#include <xmmintrin.h> // Header pour SSE

int main() {
  float a1[4] __attribute__ ((aligned (16)))
                          = {1.4, 2.5, 3.6, 4.8};
  float a2[4] __attribute__ ((aligned (16)));
  __m128 v1, v2;
  v1 = _mm_load_ps(a1);
  _mm_store_ps(a2, v1);
  printf("%f %f %f %f\n", a2[0], a2[1], a2[2], a2[3]);
  return 0;
}
```

SSE & Intrinsics : Fonctions : Opérations arithmétiques

Opérations de base

- _mm_add_pd - (Add-Packed-Double)
  - Entrée : [A0, A1], [B0, B1]
  - Sortie : [A0 + B0, A1 + B1]
- _mm_add_ps - (Add-Packed-Single)
  - Entrée : [A0, A1, A2, A3], [B0, B1, B2, B3]
  - Sortie : [A0 + B0, A1 + B1, A2 + B2, A3 + B3]

SSE & Intrinsics : Fonctions : Opérations arithmétiques

Opérations de base

- _mm_add_epi64 - (Add-Packed-LongInt)
  - Entrée : [A0, A1], [B0, B1]
  - Sortie : [A0 + B0, A1 + B1]
- _mm_add_epi32 - (Add-Packed-Int)
  - Entrée : [A0, A1, A2, A3], [B0, B1, B2, B3]
  - Sortie : [A0 + B0, A1 + B1, A2 + B2, A3 + B3]

SSE & Intrinsics : Fonctions : Opérations arithmétiques

Opérations de base

- _mm_mul_pd - (Multiply-Packed-Double)
  - Entrée : [A0, A1], [B0, B1]
  - Sortie : [A0 * B0, A1 * B1]
- _mm_mul_ps - (Multiply-Packed-Single)
  - Entrée : [A0, A1, A2, A3], [B0, B1, B2, B3]
  - Sortie : [A0 * B0, A1 * B1, A2 * B2, A3 * B3]

SSE & Intrinsics : Fonctions : Opérations arithmétiques

Addition&Soustraction (disponibles à partir de SSE3)

- _mm_addsub_pd - (Add-Subtract-Packed-Double)
  - Entrée : [A0, A1], [B0, B1]
  - Sortie : [A0 - B0, A1 + B1]
- _mm_addsub_ps - (Add-Subtract-Packed-Single)
  - Entrée : [A0, A1, A2, A3], [B0, B1, B2, B3]
  - Sortie : [A0 - B0, A1 + B1, A2 - B2, A3 + B3]

SSE & Intrinsics : Fonctions : Opérations arithmétiques

Opérations horizontales (disponibles à partir de SSE3)

- _mm_hadd_pd - (Horizontal-Add-Packed-Double)
  - Entrée : [A0, A1], [B0, B1]
  - Sortie : [B0 + B1, A0 + A1]
- _mm_hadd_ps (Horizontal-Add-Packed-Single)
  - Entrée : [A0, A1, A2, A3], [B0, B1, B2, B3]
  - Sortie : [B0 + B1, B2 + B3, A0 + A1, A2 + A3]
- _mm_hsub_pd - (Horizontal-Subtract-Packed-Double)
  - Entrée : [A0, A1], [B0, B1]
  - Sortie : [A0 - A1, B0 - B1]
- _mm_hsub_ps - (Horizontal-Subtract-Packed-Single)
  - Entrée : [A0, A1, A2, A3], [B0, B1, B2, B3]
  - Sortie : [A0 - A1, A2 - A3, B0 - B1, B2 - B3]

SSE & Intrinsics : Fonctions : Opérations logiques

Quelques instructions

_mm_{and | or | xor | ...}_{ps | pd | si128}

SSE & Intrinsics : Fonctions : Tests

Quelques instructions

_mm_cmp{eq | gt | lt | ...}_{ps | pd | epi8 | epi 16 | ...}

SSE & Intrinsics : Fonctions : Réorganisation des données

Quelques instructions

_mm_shuffle_...

SSE & Intrinsics : Fonctions : Conversions

Quelques instructions

_mm_cvt...

Plan

SSE & Intrinsics : Exemples

démo

Plan

Altivec

Nomanclature

- Velocity Engine (Apple)
- VMX (IBM)

Plateformes

- PowerPC G4&5 : Apple PowerMac
- POWER6&7 : stations IBM
- Cell Broadband Engine : stations IBM & PS3
- variantes PowerPC IBM : Xbox 360, Wii, Gamecube

AltiVec

Coming soon ...

Plan

NEON

Coming soon ...