ENSSAT
LANNION

Processeurs Multi-média

Vectorisation, instructions SSE, SSE2, … AVX

Support de cours IAI3

D.Chillet

Daniel.Chillet@enssat.fr

http://cairn.enssat.fr

ENSSAT
LANNION

Université de Rennes

Vectorisation au sein des processeur

ENSSAT LANNION

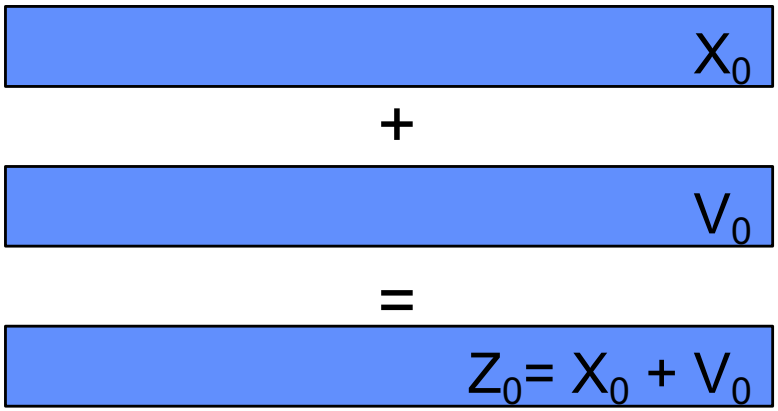
❑ Limite des processeurs

➢Calculs sur des vecteurs réalisés par des boucles

➢Z = X + V

```
for ( i = 0 ; i < 4 ; i++) {
        Z[i] = X[i] + V[i];
}
```

$$\begin{bmatrix} Z_0 \\ Z_1 \\ Z_2 \\ Z_3 \end{bmatrix} = \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} + \begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{bmatrix}$$



Vectorisation au sein des processeur

ENSSAT LANNION

❑ Limite des processeurs, suite

➢Calculs **vectoriels** réalisés sur plusieurs éléments en //

➢Z = X + V

```
int X[4] = {X3, X2, X1, X0} ;
int V[4] = {V3, V2, V1, V0} ;
int Z[4] = {Z3, Z2, Z1, Z0} ;

Z = X + Y ;
```

X_3	X_2	X_1	X_0
+	+	+	+
V_3	V_2	V_1	V_0
=	=	=	=
$Z_3 = X_3 + V_3$	$Z_2 = X_2 + V_2$	$Z_1 = X_1 + V_1$	$Z_0 = X_0 + V_0$

Vectorisation au sein des processeur

ENSSAT LANNION

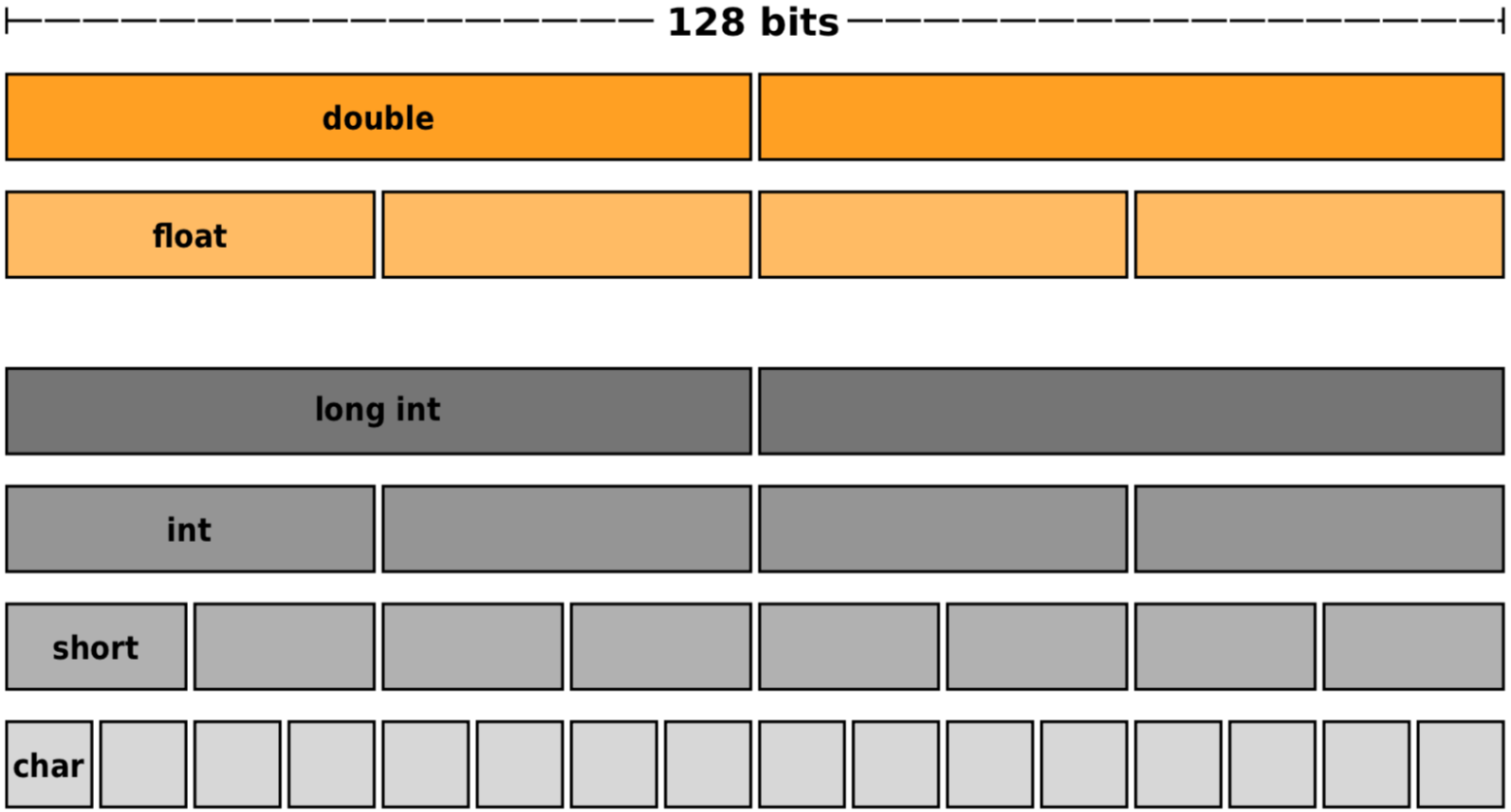
❑ Introduction des instructions vectorielles

- ➢ MMX, 1997, Intel - MultiMedia eXtensions
  - ♦ 8 registres 64 bits (partagés avec l'unité flottante)
  - ♦ 57 instructions spécifiques pour manipuler des vecteurs
  - ♦ Manipulation d'entiers uniquement
- ➢ SSE, 1999, Intel - Streaming SIMD Extensions
  - ♦ Évolutions SSE2, SSE3, SSE4
    - – SSE(N) = SSE(N-1) + nouvelles instructions
  - ♦ 8 registres 128 bits pour IA32
  - ♦ 16 registres 128 bits pour IA64
- ➢ AVX, 2008, Intel – Advanced Vector eXtension
  - ♦ Les registres passent à 256 bits
- ➢ AVX2, 2011, Intel
  - ♦ Evolution de l'AVX
- ➢ AVX2, 2013, Intel
  - ♦ Registres passent à 512 bits

Vectorisation au sein des processeur

ENSSAT LANNION

- **Types de données manipulés avec SSE**
  - __m128 : 4 floats
  - __m128d : 2 doubles
  - __m128i : entiers



Vectorisation au sein des processeur

ENSSAT LANNION

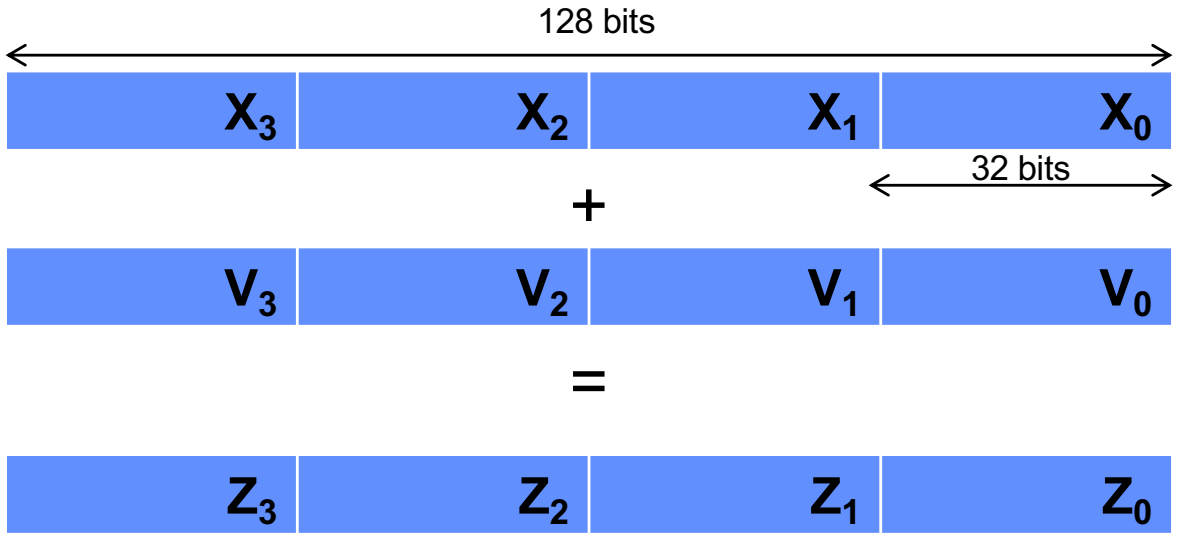
❑ Code SSE, addition float

➢Calculs **vectoriels** réalisés sur plusieurs éléments en //

➢Z = X + V ➔ SSE (float, 32 bits)

```
float X[4] = {X3, X2, X1, X0} ;
float V[4] = {V3, V2, V1, V0} ;
float  Z[4] = {Z3, Z2, Z1, Z0} ;
for ( i = 0 ; i < 4 ; i++) {
        Z[i] = X[i] + V[i];
}
```

```
__m128 X[4] = {X3, X2, X1, X0} ;
__m128 V[4] = {V3, V2, V1, V0} ;
__m128 Z[4] = {Z3, Z2, Z1, Z0} ;

Z = _mm_add_ps(X, Y);
```



Vectorisation au sein des processeur

ENSSAT LANNION

❑ Code SSE, addition double

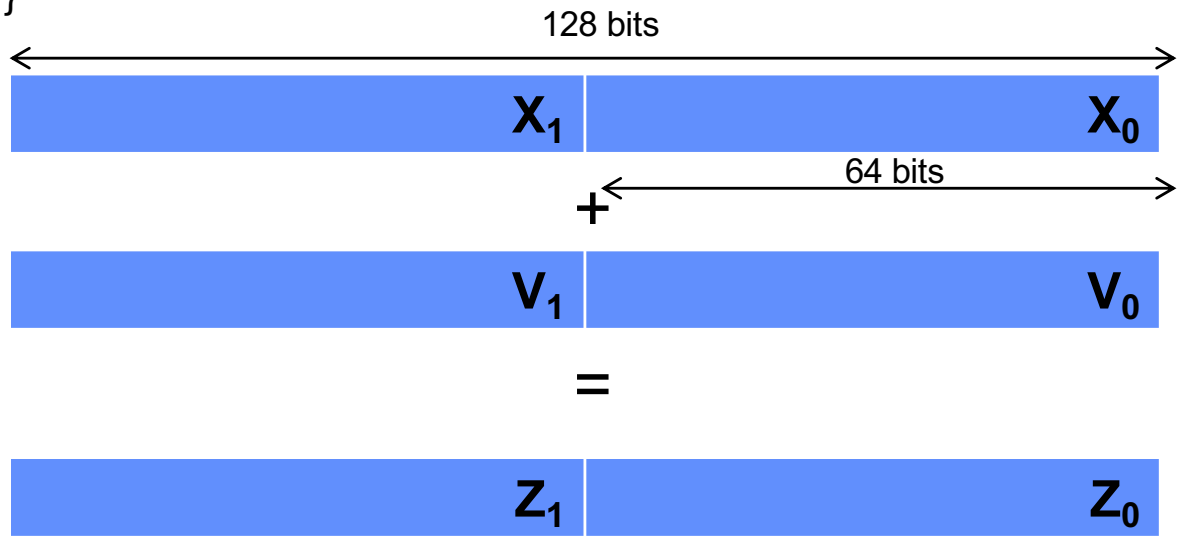
➢Calculs **vectoriels** réalisés sur plusieurs éléments en //

➢Z = X + V

➔ SSE (double 64 bits)

```
double X[2] = {X1, X0} ;
double V[2] = {V1, V0} ;
double Z[2] = {Z1, Z0} ;

for ( i = 0 ; i < 2 ; i++) {
        Z[i] = X[i] + V[i];
}
```

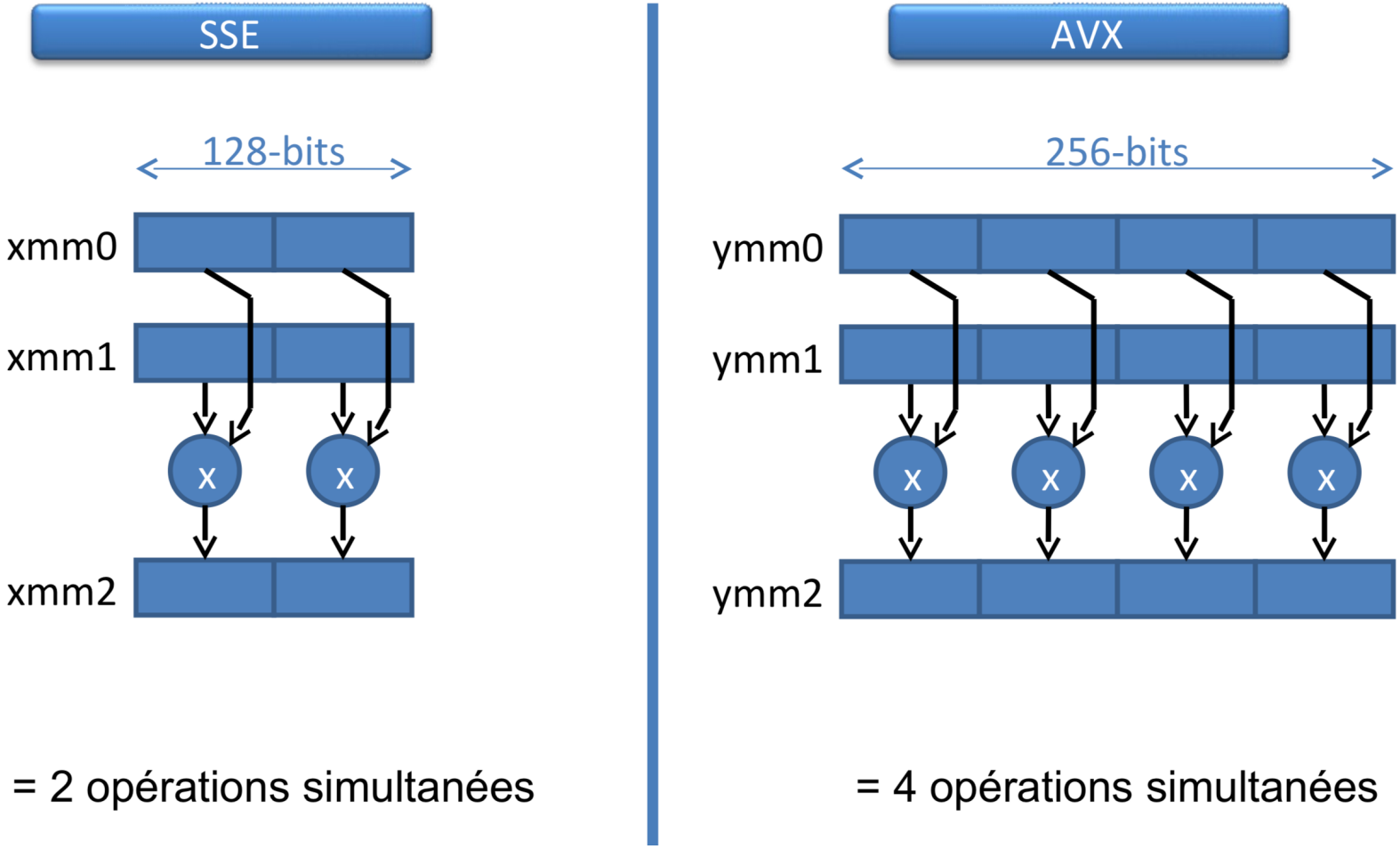
```
__m128 X[2] = {X1, X0} ;
__m128 V[2] = {V1, V0} ;
__m128 Z[2] = {Z1, Z0} ;

Z = _mm_add_pd(X, Y);
```



Vectorisation au sein des processeur

❑ Différences SSE versus AVX

➢Cas du type double



= 2 opérations simultanées

= 4 opérations simultanées

ENSSAT LANNION

Code AVX, addition float

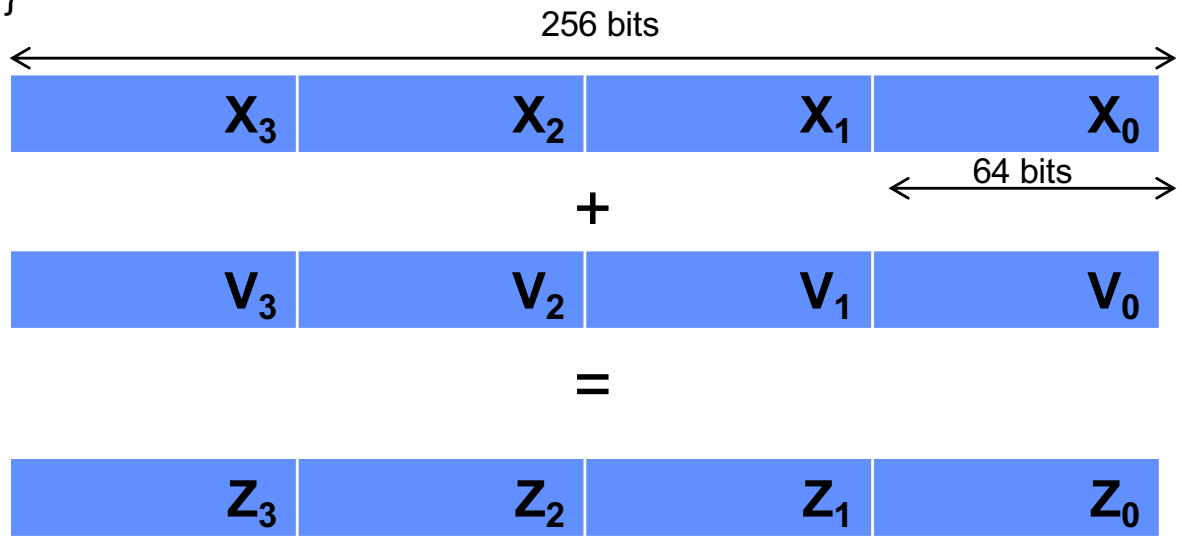
➢Calculs **vectoriels** réalisés sur plusieurs éléments en //

➢Z = X + V

➔ AVX (double 64 bits)

```
double X[4] = {X3, X2, X1, X0} ;
double V[4] = {V3, V2, V1, V0} ;
double Z[4] = {Z3, Z2, Z1, Z0} ;

for ( i = 0 ; i < 4 ; i++) {
        Z[i] = X[i] + V[i];
}
```

```
__m256 X[4] = {X3, X2, X1, X0} ;
__m256 V[4] = {V3, V2, V1, V0} ;
__m256 Z[4] = {Z3, Z2, Z1, Z0} ;

Z = _mm256_add_ps(X, Y);
```



Code AVX, addition float

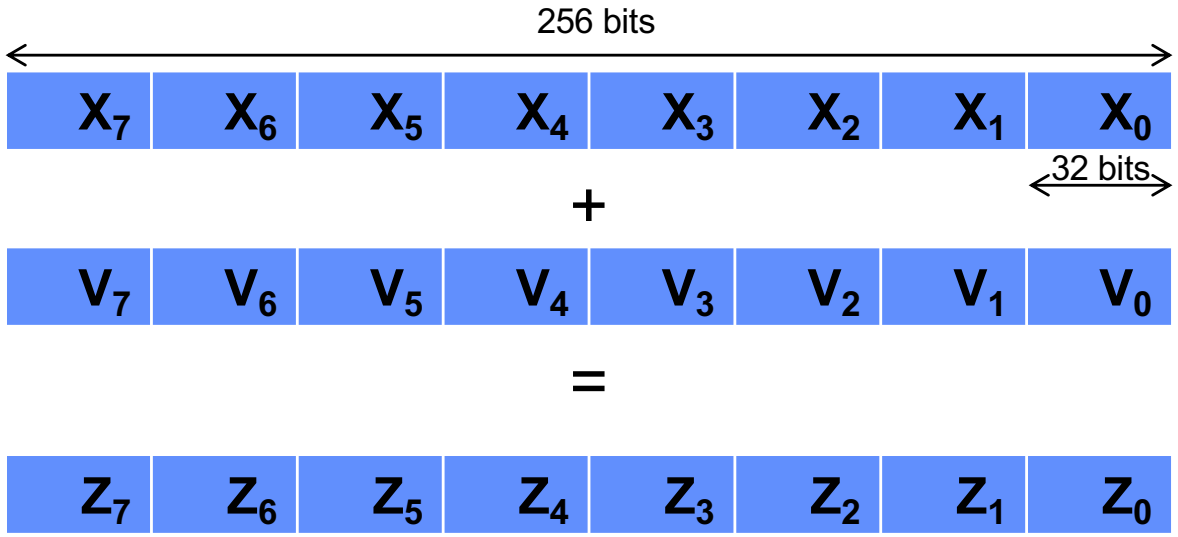
➢Calculs **vectoriels** réalisés sur plusieurs éléments en //

➢Z = X + V

➔ AVX (float 32 bits)

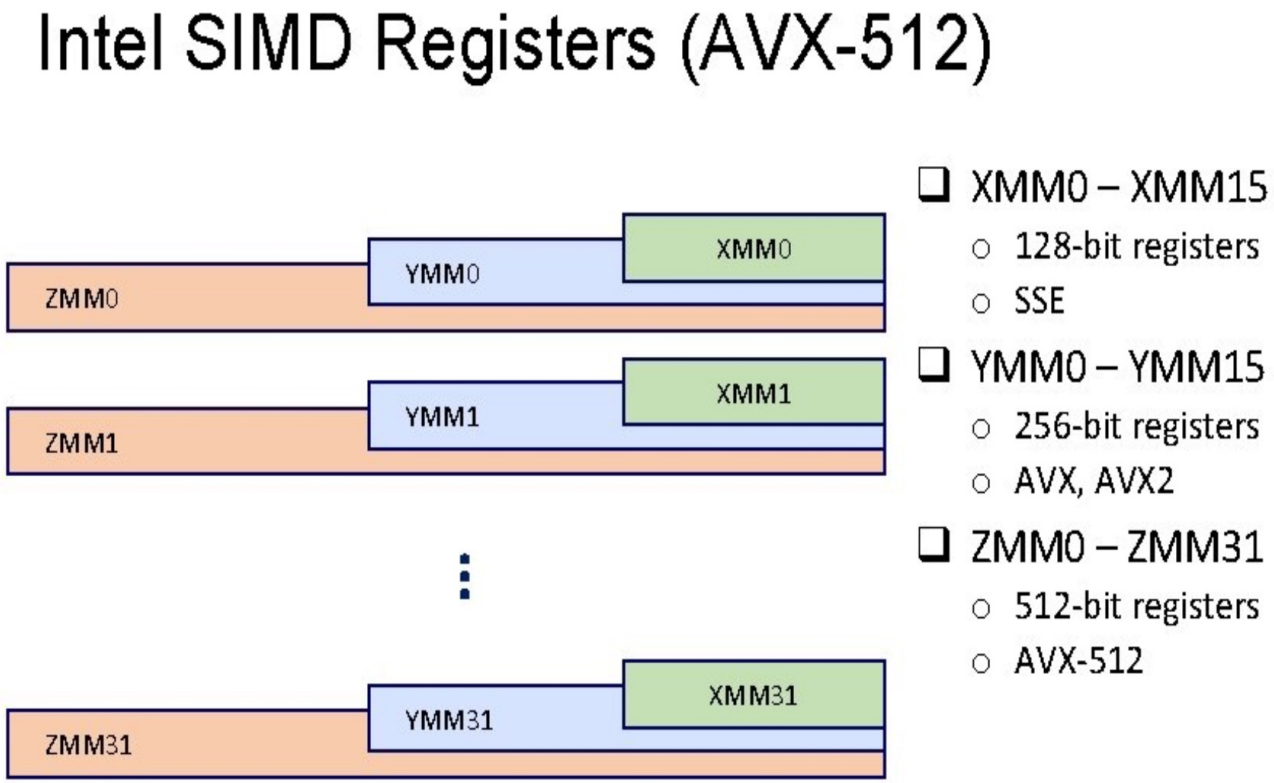
```
float X[8] = {X7, … X1, X0} ;
float V[8] = {V7, … V1, V0} ;
float Z[8] = {Z7, … Z1, Z0} ;

for ( i = 0 ; i < 2 ; i++) {
        Z[i] = X[i] + V[i];
}
```

```
__m256 X[8] = {X7, … X1, X0} ;
__m256 V[8] = {V7, … V1, V0} ;
__m256 Z[8] = {Z7, … Z1, Z0} ;

Z = _mm256_add_ps(X, Y);
```



Vectorisation au sein des processeur

❑ Banques de registres



x64 AVX-512 register scheme as extension from the x64 AVX (YMM0-YMM15) and x64 SSE (XMM0-XMM15) registers

511 256	255 128	127 0
ZMM0	YMM0	XMM0
ZMM1	YMM1	XMM1
ZMM2	YMM2	XMM2
ZMM3	YMM3	XMM3
ZMM4	YMM4	XMM4
ZMM5	YMM5	XMM5
ZMM6	YMM6	XMM6
ZMM7	YMM7	XMM7
ZMM8	YMM8	XMM8
ZMM9	YMM9	XMM9
ZMM10	YMM10	XMM10
ZMM11	YMM11	XMM11
ZMM12	YMM12	XMM12
ZMM13	YMM13	XMM13
ZMM14	YMM14	XMM14
ZMM15	YMM15	XMM15
ZMM16	YMM16	XMM16
ZMM17	YMM17	XMM17
ZMM18	YMM18	XMM18
ZMM19	YMM19	XMM19
ZMM20	YMM20	XMM20
ZMM21	YMM21	XMM21
ZMM22	YMM22	XMM22
ZMM23	YMM23	XMM23
ZMM24	YMM24	XMM24
ZMM25	YMM25	XMM25
ZMM26	YMM26	XMM26
ZMM27	YMM27	XMM27
ZMM28	YMM28	XMM28
ZMM29	YMM29	XMM29
ZMM30	YMM30	XMM30
ZMM31	YMM31	XMM31

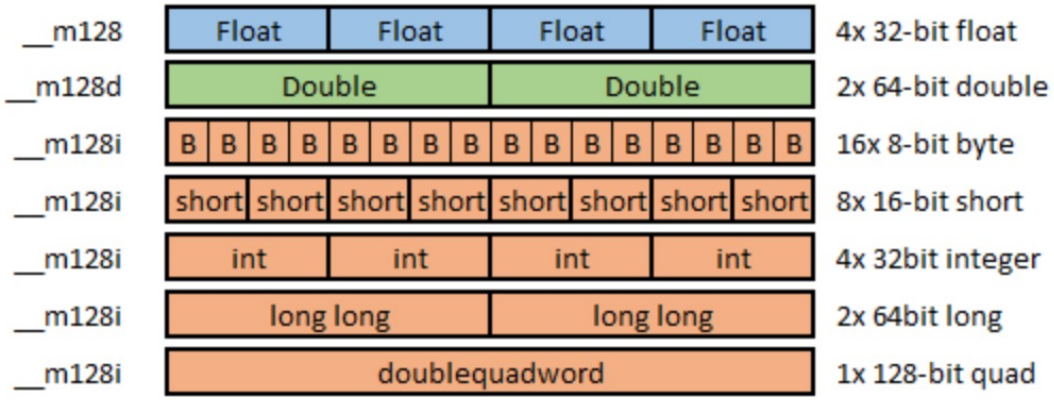
Vectorisation au sein des processeur

ENSSAT LANNION

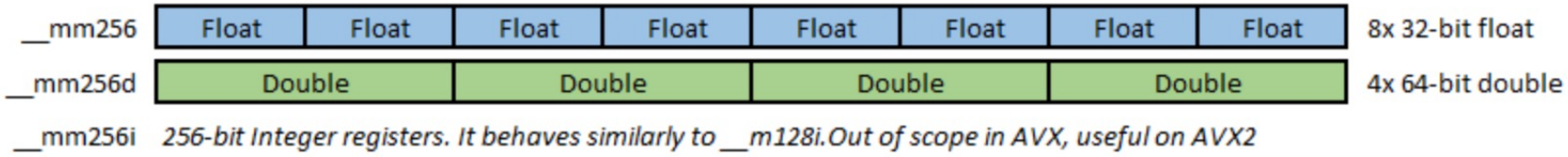
❑ Différences SSE versus AVX

- ➢ SSE types => __m128 float, __m128d double, __m128i integer
- ➢ AVX types => __m256 float, __m256d double, __m256i integer

SSE Data Types (16 XMM Registers)



AVX Data Types (16 YMM Registers)



Vectorisation au sein des processeur

ENSSAT LANNION

❑ Compatibilité type de données

Types de données manipulées par les instructions vectorielles

	MMX	SSE	SSE2	AVX	AVX2	AVX-512
__m64	Entiers 16,32 et 64bits					
__m128		Réels 32 bits				
__m128i			Entiers 16, 32 et 64 bits			
__m128d			Réels 64 bits			
__m256				Réels 32 bits		
__m256i					Entiers 8, 16, 32 et 64 bits	
__m256d				Réels 64 bits		
__m512						Réels 32 bits
__m512i						Entiers 8, 16, 32 et 64 bits
__m512d						Réels 64 bits

indisponible

disponible

Vectorisation au sein des processeur

ENSSAT LANNION

❑ Format des instructions

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

➢ Exemple

- ♦ __m128 C = _mm_add_ps(__m128 A, __m128 B)

➢ Syntaxe

- ♦ 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) / h (high) : bits de poids faible / fort
- ♦ r (reversed) : dans l'ordre inverse

ENSSAT LANNION

Format des instructions

Exemples

- __m128 C = _mm_add_pd(__m128 A, __m128 B)
- __m128 C = _mm_mul_ps(__m128 A, __m128 B)

- __m128 C = _mm_load__ps(float* p)
  - float A[4] = {1.0, 2.0, 3.0, 4.0};
  - __m128 B = _mm_load_ps(& A[0]);
  - float C[4];
  - _mm_store_ps(&C[0], B);

Vectorisation au sein des processeur

ENSSAT LANNION

❑ Additions grands vecteurs

➢Avec SSE et sur des floats 32 bits

```
float V1[1000];
float V2[1000];
float V3[1000];

__m128 VSSE1;
__m128 VSSE2;
__m128 VSSE3;

for (i=0 ; i < 1000 ; i = i + 4) {
        VSSE1 = _mm_load_ps(&V1[i]);
        VSSE2 = _mm_load_ps(&V2[i]);
        VSSE3 = _mm_add_ps(VSSE1, VSSE2);
        _mm_store_ps(&V3[i], VSSE3);
}
```

Vectorisation au sein des processeur

ENSSAT LANNION

- **Compilation**
  - ➢ Pour SSE
    - ♦ G++ fichier.c
  - ➢ Pour AVX
    - ♦ G++ fichier.c –mavx
  - ➢ Pour AVX2
    - ♦ G++ fichier.c –mavx2
- **Quels fichiers include ?**
  - ➢ SSE : xmmintrin.h
  - ➢ SSE2 : emmintrin.h
  - ➢ SSE3 : pmmintrin.h
  - ➢ SSSE3 : tmmintrin.h
  - ➢ SSE4.1 : smmintrin.h

Vectorisation au sein des processeur

ENSSAT LANNION

- **Quelles extensions dans votre processeur ?**
  - Linux : `cat /proc/cpuinfo`
  - Macosx : `sysctl machdep.cpu`
  - Windows : ?

    - machdep.cpu.features: FPU VME DE PSE TSC MSR PAE MCE CX8 APIC SEP MTRR PGE MCA CMOV PAT PSE36 CLFSH DS ACPI MMX FXSR **SSE SSE2** SS HTT TM PBE SSE3 PCLMULQDQ DTES64 MON DSCPL VMX SMX EST TM2 **SSSE3** FMA CX16 TPR PDCM **SSE4.1 SSE4.2** x2APIC MOVBE POPCNT AES PCID XSAVE OSXSAVE SEGLIM64 TSCTMR AVX1.0 RDRAND F16C
    - machdep.cpu.leaf7_features: RDWRFSGS TSC_THREAD_OFFSET SGX BMI1 HLE **AVX2** SMEP BMI2 ERMS INVPCID RTM FPU_CSDS MPX RDSEED ADX SMAP CLFSOPT IPT MDCLEAR TSXFA IBRS STIBP L1DF SSBD