Introducción

- Los procesadores evolucionan año tras año.

- Los ciclos de reloj se fueron aumentando hasta que llego un punto donde la tecnología no fue capaz de llevar el ritmo. (Sobre 2010)

- Hubo que buscar alternativas para seguir mejorando.
  - Añadir mas cores a cada procesador (Procesadores multinúcleo)
  - Paralelismo a nivel de datos (SIMD)

Introducción

- Si aumentamos la frecuencia del procesador, nuestro programa ira mas rápido, por si solo.
- Si el procesador tiene mas unidades de paralelización, nuestro programa no tiene porque mejorar. REQUIERE UN PROCESO MANUAL

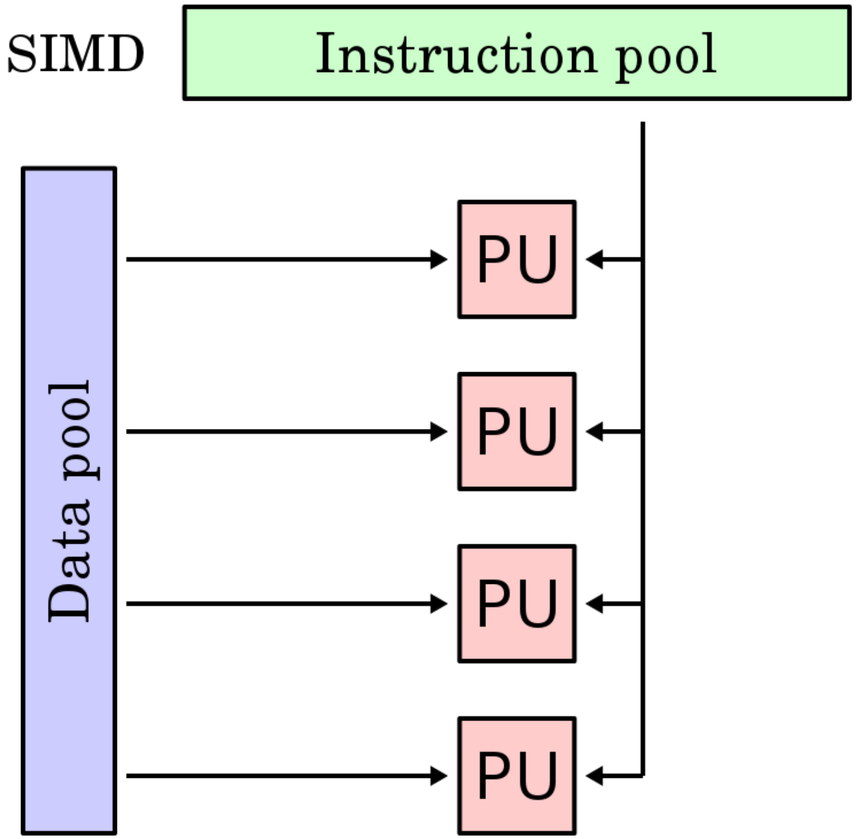
¿Quién paraleliza un programa?

- El hardware
- El compilador
- El programador

SIMD (***S**ingle **I**nstruction, **M**ultiple **D**ata*)

Mejora el rendimiento en las nuevas aplicaciones

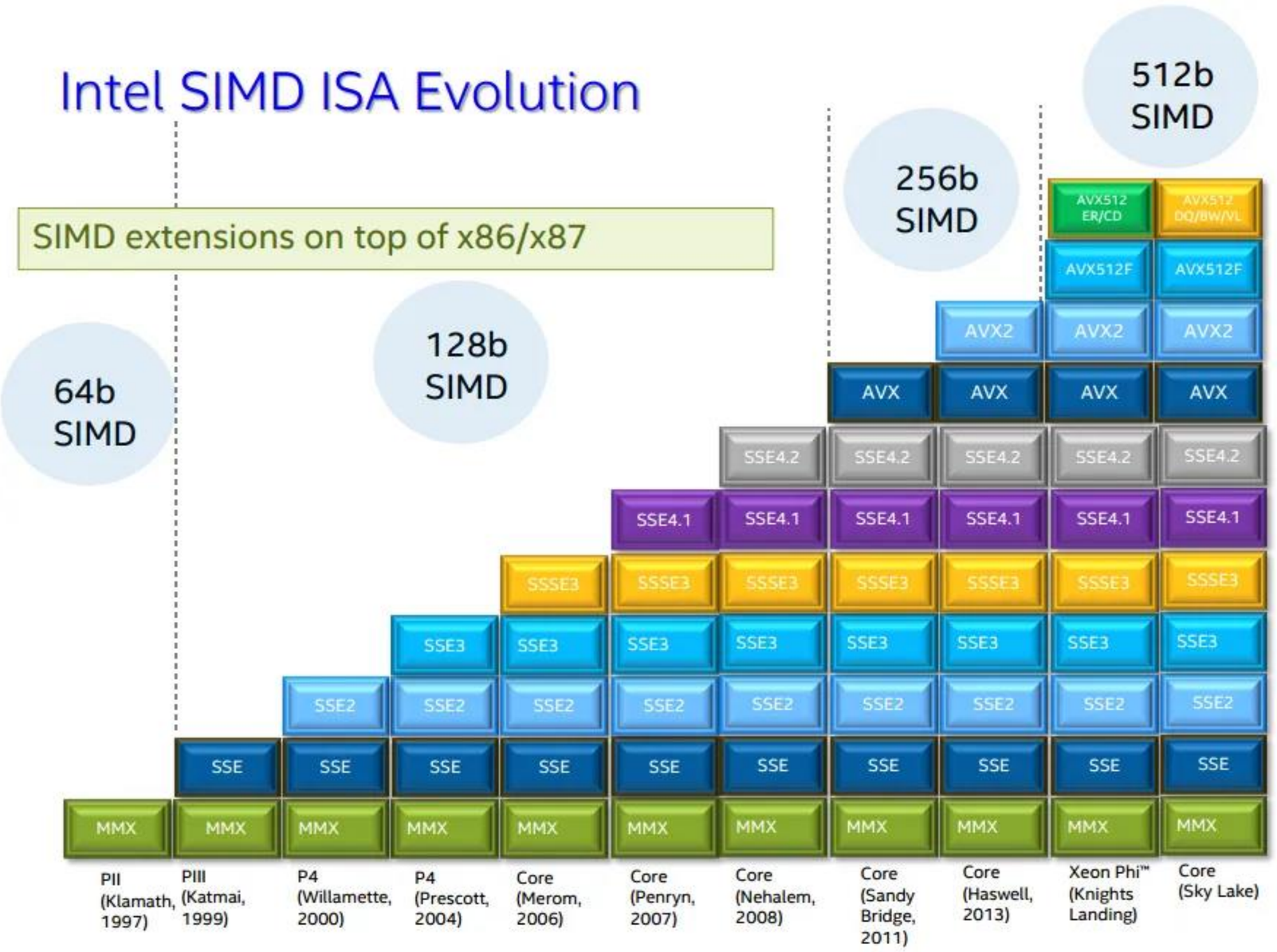
- Procesado de imagen
- Tratamiento de vídeo
- Procesamiento de audio
- Modelado 3D



Permite realizar operaciones en paralelo

- Aritméticas (ADD, SUB, MUL, DIV, SQRT, MAX, MIN, RCP, etc)
- Lógicas (AND, OR, XOR, ANDN, etc)
- Comparaciones
- Shuffle
- Manipulación de Bits
- Funciones Matemáticas
- Criptografía
- Conversión
- Y muchas más….

Extensiones vectoriales - Evolución

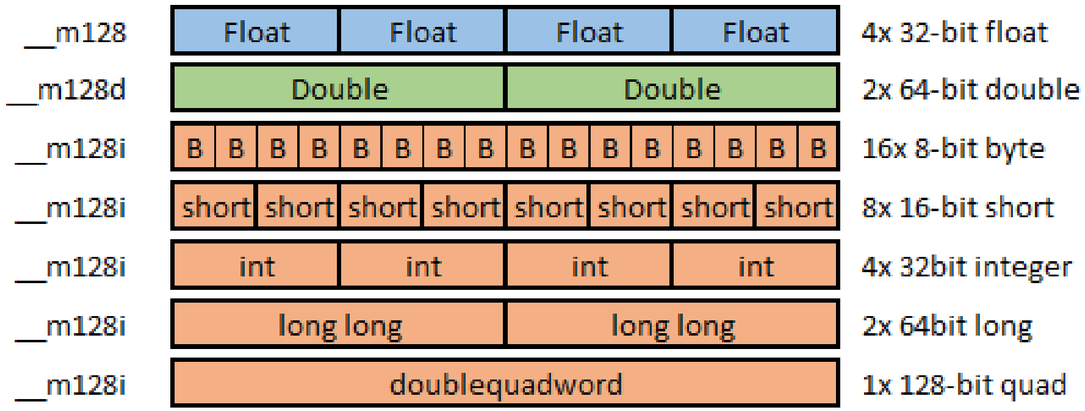


Extensiones vectoriales - Registros

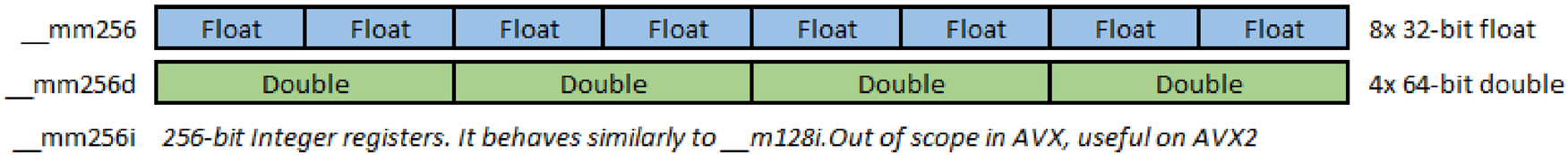
SSE y AVX tienen 16 registros cada uno. En SSE son nombrados como XMM0-XMM15, en AVX YMM0-YMM15 y en AVX512 ZMM0-ZMM31 (añade otros 16 registros).

Los XMM tienen una longitud de 128 bits, los YMM 256 bits y los ZMM 512 bits.

SSE Data Types (16 XMM Registers)

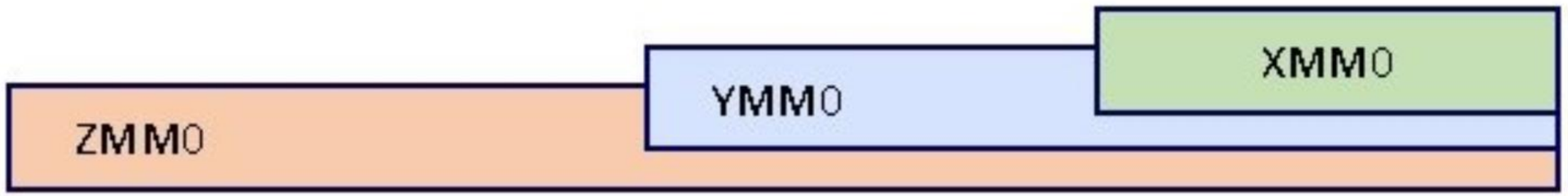


AVX Data Types (16 YMM Registers)



Extensiones vectoriales - Registros

- SSE añade tres tipos de dato __m128 __m128d __m128i de tipo float, double e integer respectivamente.
- AVX añade tres tipos de dato __m256 __m256d __m256i de tipo float, double e integer respectivamente.
- AVX512 añade tres tipos de dato __m512 __m512d __m512i de tipo float, double e integer respectivamente.



La parte inferior de los registros coincide con los registros de menor capacidad, por esa razón hay que tener cuidado al mezclar instrucciones de diferente longitud.

Extensiones vectoriales - Instrucciones

- Cada tipo de extensión vectorial tiene su propia cabecera, podemos incluirlas por separado o usar <immintrin.h> y dejar que el compilador se encargue de seleccionar las que podemos usar.
- El nombre de las instrucciones se puede separar en 3 partes
  - Longitud de la instrucción: _mm, _mm256, _mm512
  - Tipo de operación: load, store, add
  - Tipo de dato: ps(float), pd(double), epi8, epi16, epi32, epi64(integers)

_mm_load_ps: Carga 4 floats en un registro de 128 bits

_mm256_add_pd: Suma dos registros de 256 bits

_mm512_store_epi32: Guarda un registro de 512 bits como 16 enteros de 32 bits

Extensiones vectoriales - Alineamiento

- Las funciones load y store requieren que la memoria este alineada con el registro vectorial.

- El compilador no realiza ningún tipo de comprobación sobre el alineamiento de las funciones. **No alineado = segmentation fault**

- Si no es posible alinear, existen versiones loadu y storeu que pueden crear registros sobre memoria no alineada. **Requieren mas ciclos**

Extensiones vectoriales - Compilador

- Los compiladores intentan preservar la portabilidad ante todo.
- Las extensiones se activan mediante macros del preprocesador. No debemos definirlas manualmente pero si podemos usarlas para activar código mas eficiente si el hardware lo soporta.
- En gcc podemos usar gcc -dM -E - < /dev/null para comprobar que macros esta definidas.
- Es posible compilar usando cualquier extensión pero sin el hardware no se podrá ejecutar.
- Las optimizaciones –O1 a –ON de los compiladores solo hacen uso de las extensiones habilitadas.

Extensiones vectoriales - Compilador

```
[cesar.pineiro@master-bd1 ~]$ gcc -dM -E - < /dev/null | egrep "SSE|AVX"
#define __SSE2_MATH__ 1
#define __SSE_MATH__ 1
#define __SSE2__ 1
#define __SSE__ 1
```

```
[cesar.pineiro@master-bd1 ~]$ gcc -O3 -dM -E - < /dev/null | egrep "SSE|AVX"
#define __SSE2_MATH__ 1
#define __SSE_MATH__ 1
#define __SSE2__ 1
#define __SSE__ 1
```

```
[cesar.pineiro@master-bd1 ~]$ gcc -msse4.2 -dM -E - < /dev/null | egrep "SSE|AVX"
#define __SSE4_1__ 1
#define __SSE4_2__ 1
#define __SSE2_MATH__ 1
#define __SSE_MATH__ 1
#define __SSE2__ 1
#define __SSSE3__ 1
#define __SSE__ 1
#define __SSE3__ 1
```

Extensiones vectoriales - Compilador

```
[cesar.pineiro@master-bd1 ~]$ gcc -mavx2 -dM -E - < /dev/null | egrep "SSE|AVX"
#define __SSE4_1__ 1
#define __SSE4_2__ 1
#define __SSE2_MATH__ 1
#define __AVX__ 1
#define __AVX2__ 1
#define __SSE_MATH__ 1
#define __SSE2__ 1
#define __SSSE3__ 1
#define __SSE__ 1
#define __SSE3__ 1
```

```
[cesar.pineiro@master-bd1 ~]$ gcc -march=native -dM -E - < /dev/null | egrep "SSE|AVX"
#define __SSE4_1__ 1
#define __SSE4_2__ 1
#define __SSE2_MATH__ 1
#define __AVX__ 1
#define __AVX2__ 1
#define __SSE_MATH__ 1
#define __SSE2__ 1
#define __SSSE3__ 1
#define __SSE__ 1
#define __SSE3__ 1
```

Extensiones vectoriales - Ejemplos

- Bucle básico

```
for (int i = 0; i < n; i++) {
    f3[i] = f1[i] * f2[i];
}
```

- Bucle desenrollado

```
for (int i = 0; i < n; i+=4) {
    f3[i] = f1[i] * f2[i];
    f3[i + 1] = f1[i + 1] * f2[i + 1];
    f3[i + 2] = f1[i + 2] * f2[i + 2];
    f3[i + 3] = f1[i + 3] * f2[i + 3];
}
```

Extensiones vectoriales - Ejemplos

- Bucle vectorizado

```
for (int64 i = 0; i < n; i += 4) {
    __m128 a, b, c;
    a = _mm_load_ps(f1 + i);
    b = _mm_load_ps(f2 + i);
    c = _mm_mul_ps(a, b);
    _mm_store_ps(f3 + i, c);
}
```

- GCC optimizara los bucles con extensiones vectoriales con –O3 o -ftree-vectorize.

Extensiones vectoriales - Referencias

- Intel® 64 and IA-32 Architectures Software Developer's Manuals (volumes 2A and 2B). http://www.intel.com/products/processor/manuals/
- Intel IntrinsicsGuide, https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=SSSE3 (últimaconsulta 5 de marzode 2021)
- SSE Performance Programming, http://mirror.informatimago.com/next/developer.apple.com/hardwaredrivers/ve/sse.html (últimaconsulta 5 de marzode 2021)
- Intel SSE Tutorial: An Introduction to the SSE Instruction Set, http://neilkemp.us/src/sse_tutorial/sse_tutorial.html#D (últimaconsulta 5 de marzode 2021)
- Getting started with SSE programming, http://supercomputingblog.com/optimization/getting-started-with-sse-programming/ (últimaconsulta 5 de marzode 2021)