Computación de Altas Prestaciones

Seminario General 3

2022-2023

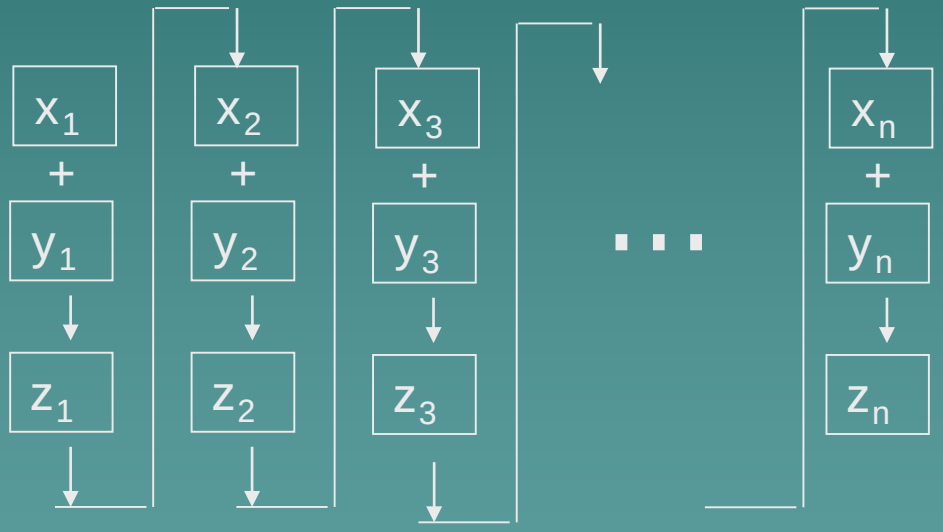
Contenido

- Vectorización e instrucciones vectoriales.
- Autovectorización.
- Experimentación con opciones de compilación.

Vectorización

Suma de dos vectores: $x, y, z \in \Re^n$

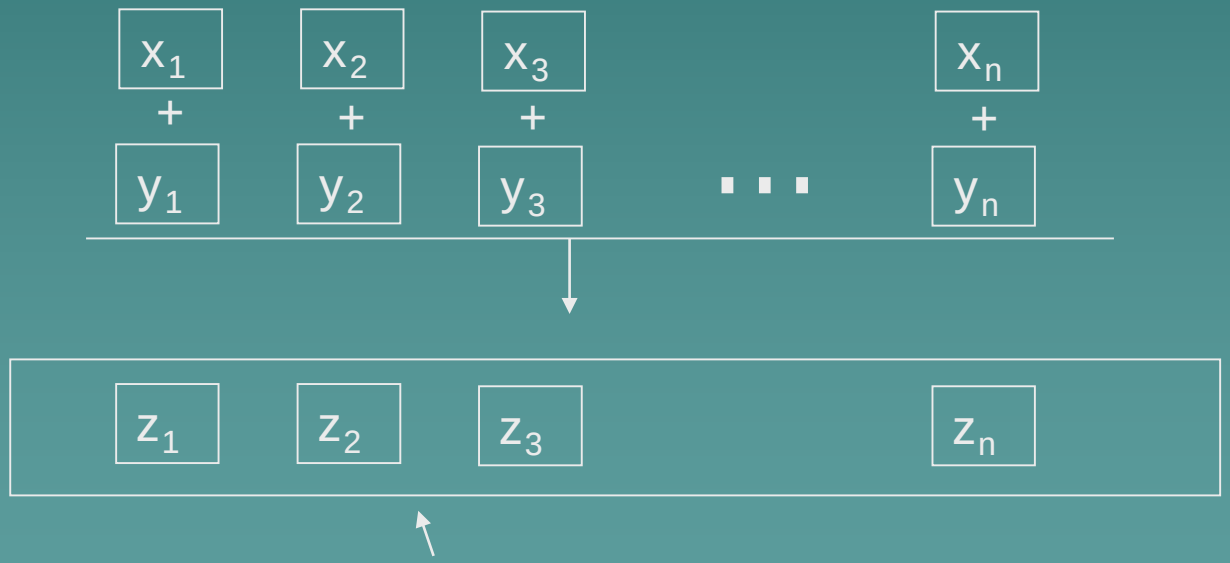
Máquina escalar



Máquinas escalares y vectoriales

Suma de dos vectores: $x, y, z \in \Re^n$

Máquina vectorial, longitud de registro vectorial mayor que **n**



Programación con instrucciones vectoriales

-En procesadores actuales tenemos diferentes conjuntos de instrucciones vectoriales:

-MMX de 64 bits
-SSE, de 128 bits
-AVX de 256 bits
-AVX de segunda generación, 512 bits (Xeon Phi nuevos)
….1024, y mas

-Cada conjunto de instrucciones tiene un conjunto de registros vectoriales asociados, a los que se puede acceder con tipos especiales de variables:

-MMX usa 16 registros de 64 bits MMX (variables de tipo _m64)
-SSE usa 16 registros de 128 bits XMM0…XMM15 (variables de tipo _m128)

Operaciones vectoriales

Se puede forzar su uso por el compilador mediante opciones de compilación (-O3, -fast). El compilador las incluirá donde detecte que es posible (y seguro) hacerlo.

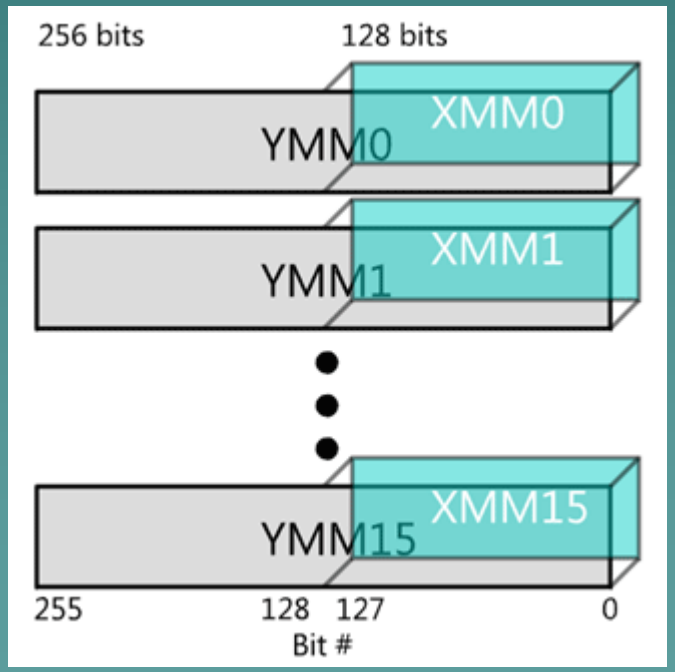
También es posible (aunque mas complicado) usarlas de forma explícita (en C o en C++)

Se pueden utilizar tanto con los compiladores de Intel, como los de Microsoft o los de gnu.

Programación con instrucciones vectoriales

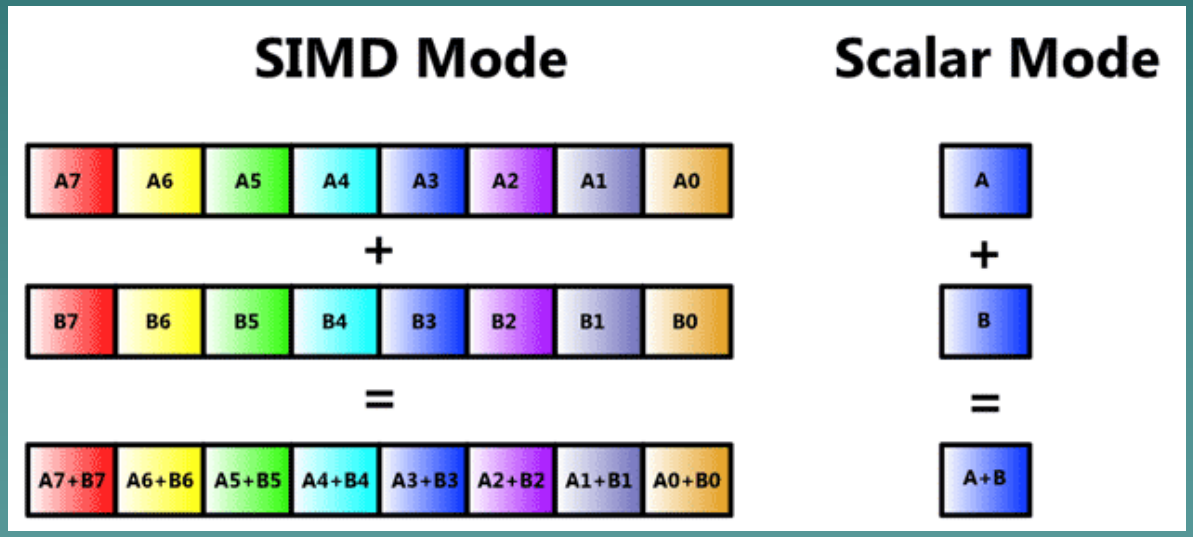
AVX usa 16 registros de 256 bits, YMM0 ... YMM15, a los que se accede ,y son manipulados, mediante variables de tipo _m256.

Los registros XMM e YMM están “aliased”



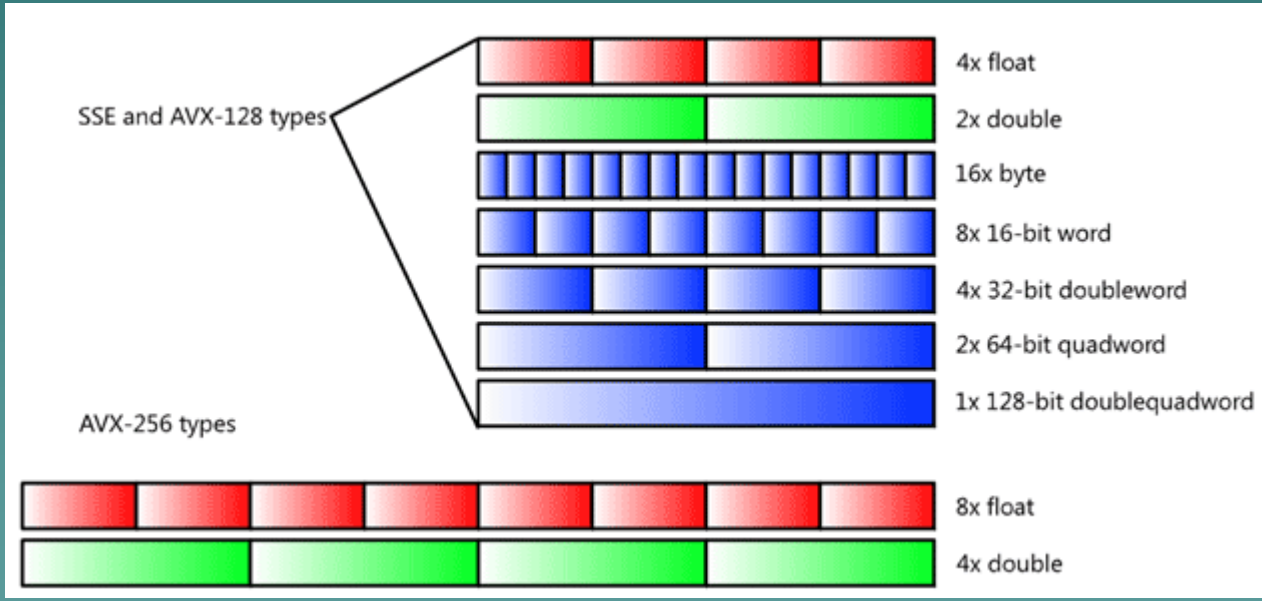
Programación con instrucciones vectoriales

Mediante estos registros, y las instrucciones asociadas podemos operar en modo SIMD



Programación con instrucciones vectoriales

Se pueden manejar diferentes tipos de datos simples



Programación con instrucciones vectoriales

Normalmente estas instrucciones las introduce el compilador (vectorización automática) en la versión ensamblador del programa, y por lo tanto no suele ser necesario programar de forma explícita.

Sin embargo, a veces el compilador no es capaz de usar de forma eficiente estas instrucciones. Usando los compiladores de Intel o Gnu, es posible usar las instrucciones vectoriales de forma explícita

Un detalle importante es que, siempre que sea posible, los datos deben estar “alineados” a 16 bytes en el caso de usar SSE de 128 bits, o a 32 bytes en el caso de usar AVX de 256 bits. AVX es mas permisivo con el uso de memoria no alineada, pero sigue siendo mas eficiente el uso de memoria correctamente alineada.

No se recomienda mezclar conjuntos de instrucciones de tipos diferentes.

Programación con instrucciones vectoriales

Podemos usar las variables de tipo _mxxx como variables “normales” o como punteros:

__m128 num1, num2; /* OJO doble barra baja */

También podemos declarar punteros de esos tipos:

__m128 *ptrsse1, *ptrsse2;

La forma de declarar vectores alineados es diferente en el compilador de gnu y en el de intel

GNU
__attribute__ ((aligned (16))) float R[100];

Intel
__declspec(align(16)) float R[100];

Programación con instrucciones vectoriales

Podemos declarar memoria alineada usando las funciones _mm_malloc, _mm_free

```
float *a;
a= _mm_malloc(10000,32) /*OJO solo una barra baja */
...
_mm_free(a)
```

Existen otras funciones que permiten declarar memoria alineada

Instrucciones vectoriales

Tenemos disponibles muchas instrucciones vectoriales:

http://www.songho.ca/misc/sse/sse.html

https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions

https://software.intel.com/sites/landingpage/IntrinsicsGuide/

...

Programación con instrucciones vectoriales (SSE)

Recorrer vectores y hacer operaciones usando instrucciones vectoriales:

Ejemplo, dados dos vectores de floats ***a*** y ***b***, de tamaño N (divisible por 4), calcular su suma y guardarla en un vector ***c***.

Código base:

```
For (i=0;i<N;i++)
   c[i]=a[i]+b[i];
```

Necesitamos la función _mm_add_ps, para sumar.

También, dependiendo del tipo de acceso, necesitamos _mm_loadu_ps y _mm_storeu_ps

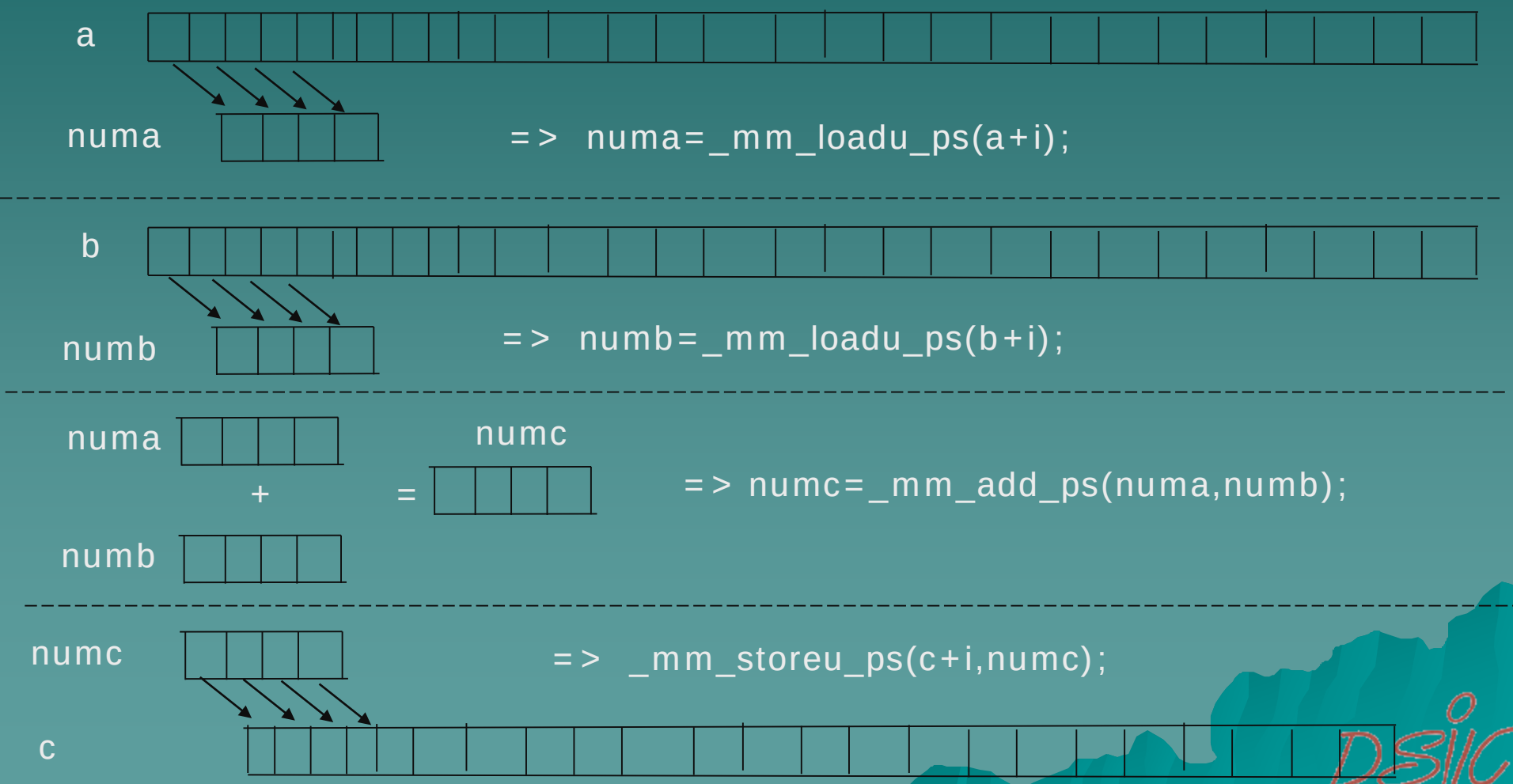
Programación con instrucciones vectoriales (SSE)

```
void suma2(float *c,float *a, float *b)
{
  int i;
 __m128 numa, numb, numc;

  for(i=0; i<SIZE; i+=4)
  {
    numa=_mm_loadu_ps(a+i);
    numb=_mm_loadu_ps(b+i);
    numc=_mm_add_ps(numa,numb);
     _mm_storeu_ps(c+i,numc);

  }
}
```

Programación con instrucciones vectoriales (SSE)



Programación con instrucciones vectoriales (SSE)

Versión con punteros

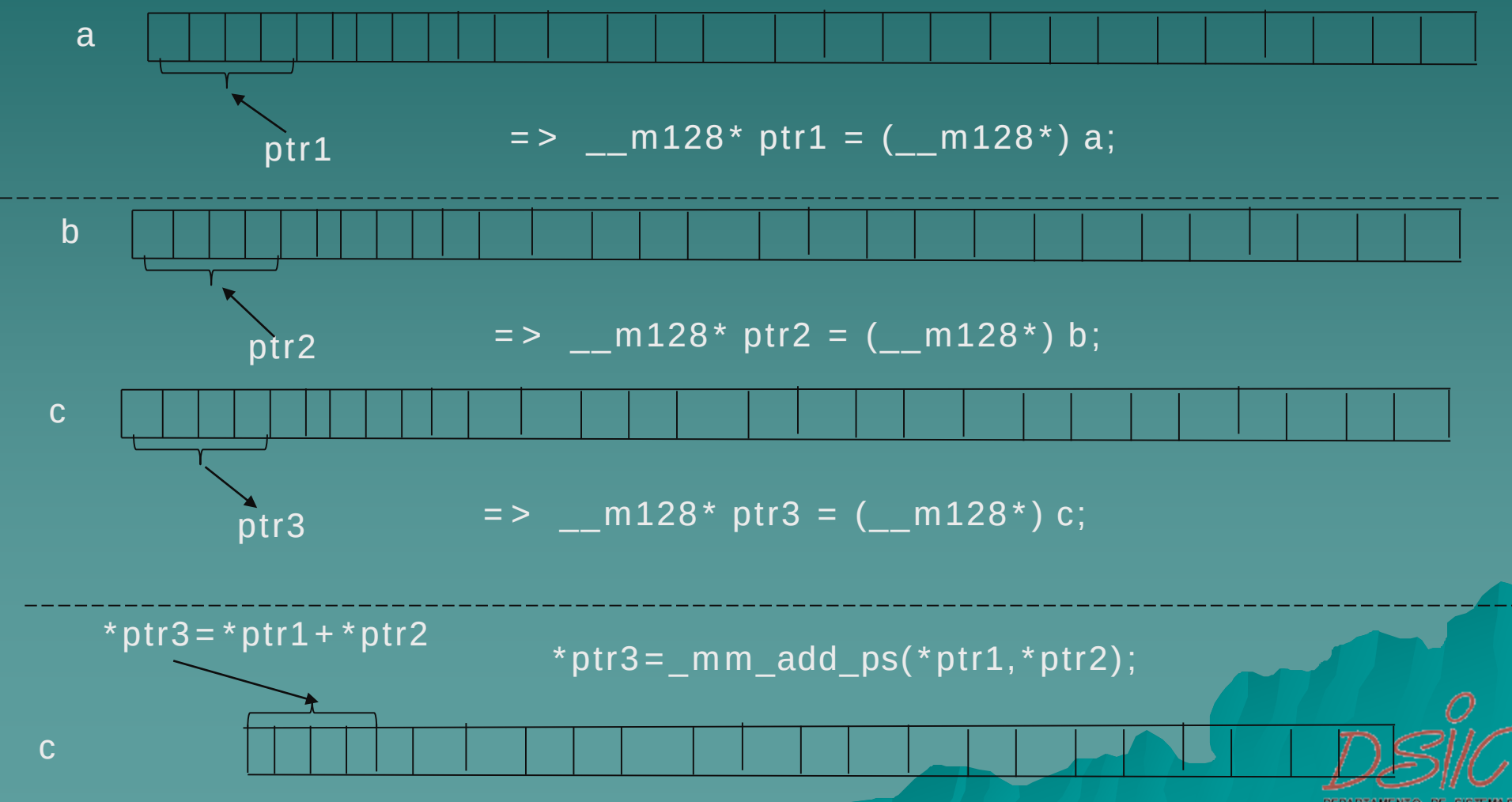
```
void suma3(float *c,float *a, float *b)
{
  int i;
 __m128* ptr1 = (__m128*) a;
 __m128* ptr2 = (__m128*) b;
 __m128* ptr3 = (__m128*) c;


  for(i=0; i<SIZE; i+=4)
  {

    *ptr3=_mm_add_ps(*ptr1,*ptr2);
    ptr1++;ptr2++;ptr3++;

  }
}
```

Programación con instrucciones vectoriales (SSE)



Programación con instrucciones vectoriales (SSE)

Otra versión con punteros

```
void suma4(float *c,float *a, float *b)
{
  int i;
 __m128 *ptr1, *ptr2, *ptr3;


  for(i=0; i<SIZE; i+=4)
  {
    ptr1=(__m128*)&a[i];
    ptr2=(__m128*)&b[i];
    ptr3=(__m128*)&c[i];
    *ptr3=_mm_add_ps(*ptr1,*ptr2);

  }
}
```

Programación con instrucciones vectoriales (AVX)

Versión AVX

```
void sumaavx(float *c,float *a, float *b)
{
  int i;
 __m256 *ptr1, *ptr2, *ptr3;


  for(i=0; i<SIZE; i+=8)
  {
    ptr1=(__m256*)&a[i];
    ptr2=(__m256*)&b[i];
    ptr3=(__m256*)&c[i];
    *ptr3=_mm256_add_ps(*ptr1,*ptr2);

  }
}
```

Programación con instrucciones vectoriales (SSE)

Ver ejemplo producto escalar, en documento pdf “C++ intrinsics”

Ejercicio : A partir del archivo saxpy_ejercicio.c, hacer una versión de la operación saxpy usando SSE.

-Saxpy (scalar a x plus y): y= ax+y

```
for i=1:n
      yi = a*x(i)+y(i)
end
```

Necesitamos las funciones _mm_set_ps1, la función _mm_add_ps, y la función _mm_mul_ps.

Introducción a Auto-vectorización

- La vectorización consiste en el desenrollado de un bucle combinado con la introducción de instrucciones SIMD
  - Nos permite ejecutar de forma simultánea la misma operación sobre varios datos.
  - Auto-vectorización se refiere a la capacidad de realizar el desarrollado y la codificación en instrucciones SIMD de forma automática.
- La opción -vec (Linux* OS) o /Qvec (Windows* OS) habilita la vectorización
  - Evidentemente, el comportamiento del compilador es mejor en procesadores Intel.
  - La vectorización puede afectar a otras opciones como /arch /0, -, -x.
- La vectorización está disponible en los niveles de optimización O2 o superiores.
  - Muchos bucles se vectorizan automáticamente, pero cuando esto no ocurre, es posible guiar al compilador añadiendo opciones o modificaciones en el código.
- Los pasos a realizar serán:
  - Medir las prestaciones base.
  - Generar un informe de vectorización
  - Mejorar las prestaciones reduciendo efectos colaterales
  - Mejorar las prestaciones con el alineamiento de los datos
  - Mejorar las prestaciones con la optimización interprocedural.

Tutorial de Autovectorización

El compilador de Intel es capaz de vectorizar bucles, pero también de “aconsejarnos” para lograr vectorizar bucles que , en primera instancia el compilador no es capaz de vectorizar.

documentación del compilador de C:
/opt/intel/composerxe/Documentation/en_US/documentation_c.htm

Se necesitan tres archivos rutinas, multiply.c, multiply.h y driver.c (en poliformat)

El problema que se explora es el producto matriz por vector

Tutorial de Autovectorización

Driver.c contiene funciones para inicializar datos, para tomar tiempos, para comprobar que el resultado está bien, y el programa principal;

Según las opciones de compilación que se elijan, se ejecuta una versión con llamada a función o una versión sin llamada a función.

Multiply.c contiene una función para hacer el producto matriz por vector, y multiply.h la cabecera.

El tutorial se puede seguir directamente (abriéndolo en knights o gpu2) o se puede seguir con las transparencias

Medir las prestaciones de Base

- Copiar los ejemplos en un directorio local (desde poliformaT)
  - Driver.c
  - Multiply.c
  - Multiply.h
- Compilar adecuadamente el ejemplo:

```
$ icc -O1 -std=c99 -DNOFUNCCALL Multiply.c Driver.c -o MatVector
```

- Ejecutar MatVector y registrar el tiempo de la ejecución
  - Esta será el tiempo base para las mediciones.

```
$ ./MatVector
```

- Este ejemplo utiliza un vector de longitud variable, (VLA), y por tanto debe compilarse con la opción -std=c99 option

Generar un Informe de Vectorización

- Un informe de vectorización indica las partes del código que han podido ser vectorizadas y en caso negativo, muestra las razones.
- Se deberá utilizar la opción –O2 para que el compilador genere el informe:

```
$ icc -std=c99 -O2 -D NOFUNCCALL -qopt-report=1 -qopt-report-phase=vec
Multiply.c Driver.c -o MatVector
```

- Ejecutar y registrar el Nuevo tiempo de ejecución.
  - La mejora es debida a la autovectorización del bucle en la línea 150.

```
Driver.c(150) (col. 4): remark: LOOP WAS VECTORIZED.
Driver.c(164) (col. 2): remark: LOOP WAS VECTORIZED.
Driver.c(81) (col. 2): remark: LOOP WAS VECTORIZED.
```

driver.c

```
76 void printsum(int length, FTYPE ans[]) {
77        // Doesn't print the whole matrix - Just a very simple Checksum
78        int i;
79        double sum=0.0;
80
81        for (i=0;i<length;i++) sum+=ans[i];
82
83        printf("Sum of result = %f\n", sum);
84 }

...

143  //start timing the matrix multiply code
144        startTime = clock_it();
145        for (k = 0;k < REPEATNTIMES;k++) {
146 #ifdef NOFUNCCALL
147        int i, j;
148                for (i = 0; i < size1; i++) {
149                        b[i] = 0;
150                        for (j = 0;j < size2; j++) {
151                                b[i] += a[i][j] * x[j];
152                        }
153                }
154 #else
…
164        printsum(COL,b);
```

Driver.c(81) (col. 2): remark: LOOP WAS VECTORIZED.

Driver.c(150) (col. 4): remark: LOOP WAS VECTORIZED.

Driver.c(164) (col. 2): remark: LOOP WAS VECTORIZED.

Generar un informe de Vectorización

- La opción –vec-report2 retorna una lista que incluye también los bucles que no han podido ser vectorizados y la razón por ello.
  - icc -std=c99 –D NOFUNCCALL –qopt-report=2 Multiply.c Driver.c -o MatVector
- El informe de vectorización indica que el bucle en la línea 45 de “Multiply.c” no ha podido ser vectorizado porque n es el más interno.
- En la línea 55 hay un bucle interno que no puede ser vectorizado por diversas razones.

```
Multiply.c(45) (col. 2): remark: loop was not vectorized: not inner loop.
Multiply.c(55) (col. 3): remark: loop was not vectorized: existence of vector dependence.
Multiply.c(55) (col. 3): remark: loop skipped: multiversioned.
Driver.c(140) (col. 2): remark: loop was not vectorized: not inner loop.
Driver.c(140) (col. 2): remark: loop was not vectorized: vectorization possible but seems inefficient.
Driver.c(141) (col. 2): remark: loop was not vectorized: vectorization possible but seems inefficient.
Driver.c(145) (col. 2): remark: loop was not vectorized: not inner loop.
Driver.c(148) (col. 3): remark: loop was not vectorized: not inner loop.
Driver.c(150) (col. 4): remark: LOOP WAS VECTORIZED.
Driver.c(164) (col. 2): remark: LOOP WAS VECTORIZED.
Driver.c(81) (col. 2): remark: LOOP WAS VECTORIZED.
Driver.c(69) (col. 2): remark: loop was not vectorized: vectorization possible but seems inefficient.
Driver.c(54) (col. 2): remark: loop was not vectorized: not inner loop.
Driver.c(55) (col. 3): remark: loop was not vectorized: vectorization possible but seems inefficient.
```

Multiply.c

```
37 #ifdef NOALIAS
38 void matvec(int size1, int size2, FTYPE a[][size2], FTYPE b[restrict], FTYPE x[])
39 #else
40 void matvec(int size1, int size2, FTYPE a[][size2], FTYPE b[], FTYPE x[])
41 #endif
42 {
43         int i, j;
44
45         for (i = 0; i < size1; i++) {
46                 b[i] = 0;
47
48 #ifdef ALIGNED
49 // The pragma vector aligned below tells the
50 // the loop is aligned on 16-byte boundary so
51 // aligned instructions to generate faster co
52 #pragma vector aligned
53 #endif
54
55                 for (j = 0;j < size2; j++) {
56                         b[i] += a[i][j] * x[j];
57                 }
58         }
59 }
```

Multiply.c(45) (col. 2): remark: loop was not vectorized: not inner loop.

Multiply.c(55) (col. 3): remark: loop was not vectorized: existence of vector dependence.
Multiply.c(55) (col. 3): remark: loop skipped: multiversioned.

Mejora de prestaciones, desambiguación de punteros

- Dos punteros son ambiguos si apuntan a la misma posición de memoria
  - Esto puede impedir optimizaciones, ya que puede implicar dependencias entre iteraciones, ya que las dependencias pueden implicar elementos en la misma operación vectorizada.
- Si sabemos con seguridad que no se dan estos solapes, podemos compilar con la opción NOALIAS que active el calificador “restrict” que informa de que el puntero no apunta a un área que sea apuntada por otro puntero.
- El calificador restrict require el uso de la opción -std=c99.
  - icc -std=c99 –qopt-report=2 –D NOALIAS Multiply.c Driver.c -o MatVector
- Esta compilación condicional reemplaza el bucle del programa principal con la llamada a la función.
  - Ejecutar Matvector y tomar tiempos.

Mejora de prestaciones por el alineamiento de datos

- El vectorizador puede generar un código más rápido cuando trabaja con datos alineados.
- Alineando los vectores a, b y x en Driver.c a nivel de 16-bytes permitimos incluir instrucciones de carga más eficientes.
  - Incluir la directive ALIGNED modificacrá las declaraciones de a, b, y x en Driver.c con la palabra clave __attribute:
  - float array[30] __attribute((aligned(base, [offset])));
  - Esta instrucción fuerza al compilador a crear un array que está alineado en el offset indicado con respecto a la dirección base (0)
  - FTYPE a[ROW][COLWIDTH] __attribute((aligned(16)));

Mejora de prestaciones por el alineamiento de datos

- Adicionalmente, la longitud de fila de la matriz a tiene que estar alineada a múltiplos de 16 bytes
  - Además indicamos al compilador que todos los arrays están alineados mediante #pragma vector aligned.
- Si se pretendiera utilizer el Nuevo set de instrucciones de Intel® AVX, deberíamos alinear a 32-bytes.
- Recompila el programa tras añadir el macro ALIGNED
  - icc -std=c99 –qopt-report=2 –D NOALIAS –D ALIGNED Multiply.c Driver.c -o MatVector

Mejora de las prestaciones con la optimización interprocedural

- El compilador es capaz de hacer optimizaciones adicionales si es capaz de optimizar más allá de los límites de las llamadas a funciones
  - Esto puede implicar, pero no limitarse, al uso de funciones en linea.
  - Se activa mediante la opción –ipo.
- Recompilar el programa utilizando la opción –ipo usando la optimización interprocedural

  $ icc -std=c99 –qopt-report=2 –D NOALIAS –D ALIGNED -ipo Multiply.c Driver.c -o MatVector

Flags para obtener informes de vectorización en gcc

- https://www.codingame.com/playgrounds/283/sse-avx-vectorization/autovectorization

- En muchos casos con gcc –O3 se vectoriza, pero no con el mejor juego de instrucciones vectoriales disponibles. Flags posibles: -mavx, -mavx512f,
- -march=native

Computación de Altas Prestaciones

-Optimización mediante flags de compilación

Experimentación con Opciones de compilación

1) Entrar en usuario DSIC, Linux, knights o gpu

2) Vamos a hacer pruebas con el producto matriz por matriz, de tamaño 2000 x 2000. Archivo prueba_producto1.c, hay que compilar también en la misma línea ctimer.c

3) Cada core de knights o de gpu ejecuta instrucciones vectoriales.

4) Los compiladores de C disponibles son gcc e icc (OneApi).

5) Probamos con gcc y con icc -O1, -O2, -O3).

Autooptimización

Probar la opción de autooptimización con icc:

- -Compilar con –prof-gen
- -Ejecutar
- -Compilar con –prof-use
- -Ejecutar

Para este problema, no se obtiene gran rendimiento.

Experimentación con Opciones de compilación

Prueba las técnicas del tutorial de autovectorización con tu subrutina de producto de matrices: -vec-report, alineación de memoria (aunque nuestra forma de reservar memoria es diferente)

-Probar opciones en subrutina de producto de matrices (combinadas con –O3) (Quitar antes paralelización OpenMP)

-guide –parallel

-parallel

Profilers

Profiler de GNU (gprof)

-Compilar con icc , –O3, y –pg (genera información de "profile"), ejecutar, ejecutar "gprof"

-> Probarlo tanto sin –parallel como con –parallel

Profiler de Intel (vtune)

-Arrancar vtune (ojo, es necesario que el comando ssh para conectar con knights sea: ssh –X knights.dsic.upv.es)

/opt/intel/vtune_amplif.../bin64/amplxe-gui