

Laboratorio Sesión 10: Instrucciones SIMD

Objetivo

El objetivo de la sesión es observar el uso de las instrucciones SIMD (Single Instruction Multiple Data) y su influencia en el rendimiento de los programas.

El formato de imagen pgm

En general los formatos de almacenamiento de imágenes se caracterizan por componerse de una cabecera con información sobre la misma seguida de una ristra (vector o matriz) de valores que representan los colores de la imagen. Muchos de ellos, además, incorporan técnicas de compresión que permiten que los datos ocupen menos. En esta práctica usaremos el formato de imagen pgm que contiene 4 valores de inicialización seguidos del valor de intensidad de gris de todos los puntos de la imagen. Es un formato muy simple que se puede leer, escribir y procesar con mucha facilidad sin tener que preocuparnos de cómo leer o guardar los datos.

Una vez los datos se han cargado en memoria, procesar la imagen consiste básicamente en aplicar operaciones a los valores que contiene para, por ejemplo, mejorar la visualización. En esta práctica en concreto vamos a descubrir una "marca de agua" que hay en la imagen de entrada `in.pgm` y que consiste en una palabra "secreta" escondida en la imagen. Para ello (y dado que esta es una marca de agua muy simple) restaremos la información de la imagen del fichero de la información de la imagen original `original.pgm` y a continuación resaltaremos el resultado para poder ver sin problemas la imagen escondida.

Las instrucciones SSE

Las instrucciones SIMD (Single Instruction Multiple Data) surgieron como una forma de aumentar la capacidad de proceso de los procesadores escalares. En entornos como el multimedia, donde tenemos que procesar una gran cantidad de datos pequeños (uno o dos bytes) todos de la misma forma, resulta muy útil aprovechar todos los bits que es capaz de procesar a la vez el procesador para realizar varias operaciones en paralelo. Como gran ventaja, estas instrucciones pueden procesar hasta 16 datos en paralelo (16 datos de un byte guardados de forma consecutiva en un registro de 128 bits), prácticamente a la misma velocidad que se procesa un registro escalar "normal" que solo contiene un dato (típicamente de 64 bits y que, por tanto, desaprovecha hasta 56 bits si almacena un byte). Como contrapartida, a veces hay que realizar muchos movimientos de datos para conseguir tener todos los datos ordenados de la forma en la que pueden procesarlos las operaciones o no tenemos la operación que se ajusta al algoritmo que queremos implementar.

En esta práctica utilizaremos las instrucciones de la extensión SSE que operan con los registros `xmm`, en concreto entre otras, las operaciones `psubb`, `pcmpgtb`, `movdqa` y `movdq`.

Estudio Previo

1. Buscad para qué sirven y qué operandos admiten las instrucciones `psubb`, `pcmpgtb`, `movdqa`, `movdq` y `emms`.
2. Buscad para qué sirve y cómo se usa en C la propiedad `__attribute__` y el atributo `aligned`.
3. Programad en ensamblador sin usar instrucciones SSE una versión de la rutina que hay en `Procesar.c` procurando hacerla lo más rápida posible (1 solo bucle, acceso secuencial...):

```
void procesar(unsigned char *mata, unsigned char *matb,
              unsigned char *matc, int n) {
    int i, j;

    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            matc[i*n+j]=(mata[i*n+j] - matb[i*n+j]);
            if (matc[i*n+j]>0)
                matc[i*n+j]=255;
            else
                matc[i*n+j]=0;
        }
    }
}
```

4. Explicad como se puede cargar un valor inmediato en un registro `xmm` usando la instrucción `movdqu`.
5. Programad en ensamblador una versión SIMD de la rutina que hay en `Procesar.c` usando las instrucciones `psubb`, `pcmpgtb` y `movdqu`.
6. Escribid un código en ensamblador que, a partir de un valor almacenado en un registro, averigüe si es múltiplo de 16.

Trabajo a realizar durante la Práctica

1. Compilad y ejecutad el programa `Transformar.c` junto con la implementación de la rutina `procesar` que hay en el fichero `Procesar.c`. Averiguad cuál es el tiempo de ejecución del programa y de todas las ejecuciones de la rutina `procesar`. Calculad cuál sería la ganancia máxima del programa si la rutina `procesar` se ejecutara de forma instantánea. Averiguad cuál es la imagen que se obtiene.
2. Implementad vuestra versión mejorada en ensamblador de la rutina `procesar` en el fichero `Procesar_asm.s`. Compilad y ejecutad el programa `Transformar.c` junto a este, comprobad que la imagen de salida es correcta y medid el tiempo de ejecución del programa y de todas las ejecuciones de la rutina `procesar`. Averiguad cuál es el speedup de la rutina obtenido respecto a la versión original y calculad de nuevo cuál sería la ganancia máxima del programa si la rutina `procesar` se ejecutara de forma instantánea. Cuando funcione entregad el fichero `Procesar_asm.s` en el Racó de la asignatura.
3. Implementad vuestra versión con instrucciones SIMD de la rutina `procesar` en el fichero `Procesar_unal.s`. Compilad y ejecutad el programa `Transformar.c` junto a este, comprobad que la imagen de salida es correcta y medid el tiempo de ejecución del programa y de todas las ejecuciones de la rutina `procesar`. Averiguad cuál es el speedup de la rutina obtenido respecto a la versión original y calculad de nuevo cuál sería la ganancia máxima del programa si la rutina `procesar` se ejecutara de forma instantánea. Cuando funcione entregad el fichero `Procesar_unal.s` en el Racó de la asignatura.

Nota: Recordad que si necesitáis declarar una variable en ensamblador podéis hacerlo con la directiva `.data`. Por ejemplo, para declarar una variable de 128 bits que contenga un 3 en cada uno de sus 16 bytes podríais hacer:

nombrevARIABLE: .byte 3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3

4. Copiad el fichero `Procesar_unal.s` en el fichero `Procesar_align.s` cambiando las instrucciones `movdqu` por `movdqqa`. Probad a compilar y ejecutar. Veréis que probablemente os da un mensaje de error.
A continuación compilad de nuevo forzando la alineación de las matrices (en el programa `Transformar.c`), comprobad que la imagen de salida es correcta y medid el

tiempo de ejecución del programa y de todas las ejecuciones de la rutina procesar. Averiguad cuál es el speedup de la rutina obtenido respecto a la versión original y calculad de nuevo cuál sería la ganancia máxima del programa si la rutina procesar se ejecutara de forma instantánea. Cuando funcione entregad el fichero Procesar_align.s en el Racó de la asignatura.

Nota: Puede ser que tengáis que alinear también las variables declaradas en ensamblador usando la directiva .align.

5. Realizad una nueva versión de la rutina procesar que ejecute vuestro código con las instrucciones movdqu o movdq a en función de la alineación de los datos que recibe como parámetro y llamadlo Procesar_dual.s. Comprobad su funcionamiento. Cuando funcione entregad el fichero Procesar_dual.s en el Racó de la asignatura.

testl op1, op2 # op1 & op2 ZF = 1 si op1 = op2
je # salta si ZF = 0, es decir si op1 ≠ op2
jne # salta si ZF = 1, es decir si op1 = op2

Nombre: _____

Grupo: _____

Nombre: _____

Hoja de respuesta al Estudio Previo

1. Explicad para qué sirven y qué operandos admiten las instrucciones:

psubb

Resta un paquete de enteros. La instrucción:

psubb mm, mm/m64

psubb xmm1, xmm2/m128

pcmpgtb

Compara paquetes de enteros con signo para valores mayor que. La instrucción:

pcmpgtb mm, mm/m64

pcmpgtb xmm1, xmm2/m128

movdqqa

Mueve valores enteros empaquetados alineados de xmm2/m128 a xmm1 o

Viceversa. La instrucción: movqqa xmm1, xmm2/m128

movqqa xmm2/m128, xmm1

movdqqu

Mueve valores enteros empaquetados no alineados de xmm2/m128 a xmm1 o

Viceversa. La instrucción: movqqu xmm1, xmm2/m128

movqqu xmm2/m128, xmm1

emms

Vacia los registros MMX y cambia el valor de la palabra de punto flotante a vacío.

La instrucción: emms

2. La propiedad attribute y el atributo aligned sirven para:

attribute indica atributos especiales al declararlos y

aligned indica el número de bytes que una variable debe estar especificada.

3. Programad en ensamblador una versión de la rutina que hay en Procesar.c procurando hacerla lo más rápida posible.

<pre> procesar: pushl %ebp movl %esp, %ebp Subl \$16, %ebp pushl %ebx pushl %esi movl 8(%ebp), %eax movl 12(%ebp), %ebx movl 16(%ebp), %ecx movl 20(%ebp), %esi imul %esi, %esi addl %ecx, %esi </pre>	<pre> for: cmpl %ecx, %esi jle end-for movb (%eax), %dl subb (%ebx), %dl if: cmpb \$0, %dl jne else movb \$255, (%eax) jmp end-if else: movb \$0, (%eax) end-if: incl %eax incl %ebx incl %ecx jmp for end-for: </pre>	<pre> popl %esi popl %ebx movl %ebp, %esp popl %ebp ret </pre>
---	--	--

4. Explicad como se puede cargar un valor inmediato en un registro xmm usando la instrucción movdqu.

Declarando en el .data una variable por ejemplo:

var: .byte 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 y moviendo esta variable a un registro xmm: movdqu var, xmm0

5. Programad en ensamblador una versión SIMD de la rutina que hay en Procesar.c.

<pre> .data comparar: .byte 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 procesar: pushl %ebp movl %esp, %ebp Subl \$16, %ebp pushl %ebx pushl %esi movl 8(%ebp), %eax movl 12(%ebp), %ebx movl 16(%ebp), %ecx movl 20(%ebp), %esi imul %esi, %esi addl %ecx, %esi </pre>	<pre> for: cmpl %ecx, %esi jle end-for movdqu (%eax), %xmm0 movdqu (%ebx), %xmm1 psubb %xmm0, %xmm1 movdqu comparar, %xmm2 pcmpegb %xmm0, %xmm2 mordq %xmm1, (%eax) addl \$16, %eax addl \$16, %ebx addl \$16, %ecx jmp for end-for: </pre>	<pre> enmms popl %esi popl %ebx movl %ebp, %esp popl %ebp ret </pre>
--	---	--

6. Escribid un código en ensamblador que a partir de un valor almacenado en un registro averigüe si es múltiplo de 16.

<pre> #Suponemos que el valor está #almacenado en el registro %eax andl \$0x0F, %eax cmpl \$0, %eax jne not-multiple ... jmp end not-multiple: ... end: </pre>	
--	--

Nombre: _____

Grupo: _____

Nombre: _____

Hoja de respuestas de la práctica

NOTA: Recordad que para compilar los programas en ensamblador 32 bits deberéis usar la opción de compilación de *gcc -m32*.

Rellenad la siguiente tabla:

Código	Tiempo ejecución	Tiempo rutina procesar	SpeedUp rutina	Ganancia potencial del programa
Original	33'75s	33'42s	—	102'27
Optimizado	7'89s	7'62s	4'38	29'2
SIMD Unaligned	0'725s	0'427s	78'26	2'43
SIMD Aligned	0'701s	0'430s	77'72	2'58

Recordad entregar los ficheros *Procesar_asm.s*, *Procesar_unal.s*, *Procesar_align.s* y *Procesar_dual.s* en el Racó de la asignatura. Debéis entregar sólo los cuatro ficheros fuentes, sin comprimir ni cambiarles el nombre, y sólo una versión por pareja de laboratorio (es indistinto que miembro de la pareja entregue).