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 normalizaremos la imagen `in`.pgm que contiene una foto tan oscura que es imposible distinguirla. Para ello reescalaremos sus valores que van entre 0 y 15 (de gris muy oscuro a negro) de forma que los pixels más claros sean blancos y avancen progresivamente hasta el negro. Esto se puede conseguir, simplemente, multiplicando todos sus valores por 16 (para acercarnos al valor máximo del formato `pgm` de 1 byte que es de 255) y es equivalente a ajustar el contraste de la imagen.

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 `paddb`, `movdqa` y `movdqu`.

Estudio Previo

1. Buscad para qué sirven y qué operandos admiten las instrucciones `paddb`, `movdqa`, `movdqu` 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, int n) {
  int i, j;

  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      matb[i*n+j]=(mata[i*n+j]*16);
    }
  }
}
```

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 `paddb` y `movdqu`.

6. Escribid un código en ensamblador que, a partir de un valor almacenado en un registro, averigüe si es multiplo 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 instrucciónes `movdqu` por `movdqa`. 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 `movdqa` 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.

Nombre: _____ Grupo: _____

Nombre: _____

Hoja de respuesta al Estudio Previo

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

`paddb`

`paddb` agrega enteros de bytes empaquetados.

`movdqa`

`movdqa` mueve un double quadword alineado.

`movdqu`

`movdqu` mueve un double quadword no alineado.

`emms`

`emms` vacía el estado de MMX.

2. La propiedad `__attribute__` y el atributo `aligned` sirven para:

`__attribute__` nos permite especificar atributos especiales al hacer una declaración.

Uno de ellos es `aligned`, que especifica a cuantos bytes debe estar alineada la variable.

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

```
    pushl %ebp
    movl %esp, %ebp
    subl $8, %esp

    pushl %ebx

    movl 8(%ebp), %eax    #eax = mata
    movl 12(%ebp), %ebx   #ebx = matb

    movl 16(%ebp), %ecx   #ecx = n
    imul %ecx, %ecx       #ecx = n^2
    addl %eax, %ecx       #ecx = @mata[n^2]
```

```
for:    cmp %eax, %ecx
        jle fifor                       #jmp if @mata[n^2] <= @mata[i]

        movb (%eax), %dl                #dl = mata[i]
        salb $4, %dl                    #dl = dl << 4
        movb %dl, (%ebx)                #matb[i] = dl

        incl %eax
        incl %ebx
        jmp for

fifor:  popl %ebx

        movl %ebp, %esp
        popl %ebp

        ret
```

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

Con la instrucción `movdqu xmm1, xmm2/m128` moveremos un double quadword no alineado de `xmm2` (que se encuentra en una posición de 128 bits de memoria) a `xmm1`.

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

```
  pushl %ebp
  movl %esp, %ebp
  subl $8, %esp
  pushl %ebx

  movl 16(%ebp), %edx         #edx = n
  imul %edx, %edx             #edx = n^2

  movl 8(%ebp), %eax #eax = @mata
  movl 12(%ebp), %ebx         #ebx = @matb

  addl %eax, %edx             #edx = @mata[n^2]
```

```
for:    cmp %eax, %edx
        jle fifor

        movdqu (%eax), %xmm0            #xmm0 = mata[0] .. mata[15]

        paddb %xmm0, %xmm0
        paddb %xmm0, %xmm0
        paddb %xmm0, %xmm0
        paddb %xmm0, %xmm0              #xmm0 = mata[0]*16..mata[15]*16

        movdqu %xmm0, (%ebx)            #matb[0]..matb[15] = mata[0]*16..mata[15]*16

        addl $16, %eax
        addl $16, %ebx

        jmp for

fifor:  popl %ebx
        movl %ebp, %esp
        popl %ebp
        ret
```

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

```
        andl 0x0000000F, %eax
        jne no_mult

mult:
        # código
        jmp fi

no_mult:
        # código

fi:
```

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	18151.14	17902.56	—	73.01
Optimizado	7669.01	7422.00	2.41	31.05
SIMD Unaligned	870.54	623.39	28.72	3.52
SIMD Aligned	868.18	622.34	28.77	3.53

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).