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.[es]condida en la imagen. Para ello (y dado que esta es una marca de agua muy simple) eliminaremos la información de la imagen del fichero (que se encuentra almacenada en los bits altos de cada pixel) y a continuación resaltaremos el bit bajo 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 `pand`, `pcmpgtb`, `movdqa` y `movdqu`.

Estudio Previo

1. Buscad para qué sirven y qué operandos admiten las instrucciones `pand`, `pcmpgtb`, `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] & 1);
      if (matb[i*n+j]>0)
        matb[i*n+j]=255;
      else
        matb[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 `pand`, `pcmpgtb` 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: Ferran Solanes Grupo: 13

Nombre: Joan Acero

Hoja de respuesta al Estudio Previo

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

pand

Destino = Destino & Source (AND lógica).
PAND mm, mm (64 bits)
PAND xmm, xmm (128 bits)

pcmpgtb

Si un elemento del operando destino es mayor que el del fuente, se pone a 1 en el destino. PCMPGTB mm, mm (64 bits)
PCMPGTB (128 bits). En este caso comparamos bytes.

movdqa

Move double quadword aligned. Can move 128, 256 or 512 bytes from memory to xmm register or vice versa.
The packet must be aligned. MOVDQA dst, src.

movdqu

Move double quadword unaligned. Can move 128, 256, 512 bytes from memory to xmm register or vice versa. The packet doesn't have to be aligned. MOVDQU r1/m, r2/m.

emms

Clears the MMX registers and sets the value of the floating point tag word to empty. Has no operands.
EMMS.

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

__attribute__ es un atributo GCC que sirve para ayudar al compilador a incorporar optimizaciones.
aligned sirve para decir a que valor se alinean los datos.

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
pushl %ebx
pushl %edi   pushl %esi

movl 8(%ebp), %eax   # eax <- @mata
movl 12(%ebp), %ebx  # ebx <- @matb
movl 16(%ebp), %edi  # edi <- n
imul %edi, %edi      # edi <- n^2
movl $0, %esi
for:
  movb (%eax, %esi), %dl
  cmpb $1, %dl
  jge else
  if:
  movb $255, (%ebx, %esi)
  jmp end_if
  else:
  movb $0, (ebx, %esi)
  end_if:
   incl %esi
  cmpl %edi, %esi
  jge end_for
  jmp for
  end_for:
popl %esi
popl %edi
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`.

movdqu no permite el uso de inmediatos como operando, por tanto o cargamos el inmediato a un registro previamente o lo guardamos a memória previamente.

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

```
pushl %ebp
movl %esp, %ebp
pushl %ebx  pushl %edi  pushl %esi

movl 8(%ebp), %eax  <- %eax <- @mata
movl 12(%ebp), %ebx <- %ebx <- @matb
movl 16(%ebp), %edi <- %edi <- n
imul %edi, %edi     <- %edi <- n^2
movl $0, %esi       <- k = n.i + j = 0
pand $0, %xmm0      <- per comparar amb registres xmm
for:
 movdqu (%eax, %esi), %xmm1 <- 16 bytes
 pand $1, %xmm1     <- %xmm1 = 1 & %xmm1

pcmpgtb %xmm0, %xmm1 <- pone a 0 los que no son mayores y a 1 los que si lo son.
movdqu %xmm1, (%ebx, %esi)
addl $16, %esi
cmpl %edi, %esi
jge end_for
jmp for
end_for:
popl %esi  popl %edi  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.

```
Si es multiplo de 16 -> ultimos 4 bits son 0

Suponemos el valor esta en el registro %eax.

 andl $0xF, %eax  <- si da zf=1 es divisible entre 16
 jne falso
verdadero:
   ...
   tratamiento caso verdadero
   ...
  jmp endif
falso:
  ...
  tratamiento caso falso
  ...
endif:
  ...
```

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			—	
Optimizado				
SIMD Unaligned				
SIMD Aligned				

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).