Trabajo Práctico 2

Organización del Computador II

Primer cuatrimestre 2015

1. Introducción

En este trabajo práctico se busca una primera aproximación al procesamiento con instrucciones que operan con mutiples datos (SIMD). El objetivo es conocer y comprender los principios de la programación con dichas instrucciones. La familia de procesadores x86-64 de Intel posee una serie de extensiones para operaciones SIMD. En un comienzo, éstas se denominaron MMX (MultiMedia eXtension), luego SSE (Streaming SIMD Extensions) y por último AVX (Advanced Vector Extensions).

En este trabajo práctico deberán desarrollar ciertas funciones utilizando el conjunto de instrucciones de SSE. Las funciones corresponden a una serie de filtros para el procesamiento de imágenes en formato BMP. Dichas imágenes serán cargadas mediante una biblioteca de funciones provista por la catedra.

El trabajo práctico consiste en implementar cada filtro en C y ASM, para luego evaluar su rendimiento. La evaluación debe ser un análisis exhaustivo de las propiedades del código ejecutado y de las circunstancias que justifican por qué una implementación funciona mejor (o peor) que otra.

2. Filtros

Una imagen BMP es una matriz de pixeles. Cada pixel está determinado por cuatro bytes: `A` = Transparencia, `B` = Azul, `G` = Verde, `R` = Rojo, dando un tamaño total de 4 bytes y almacenados en memoria como `ABGR`. Cada componente de color almacena un número entero en notación sin signo.

En todos los casos la componente `A` debe ser dejada con el mismo valor que en la imagen orginal.

2.1. Convertir YUV a RGB y RGB a YUV

Los filtros descriptos a continuación realizan la conversión entre espacios de color `RBG` y `YUV`. Este último esta comuesto por una componente de *luminancia* en dos componentes de *crominancia*. Inicialmente fue desarrollado para codificar el color para televisión analógica, ya que por compatibilidad con la ByN, el color se envia de forma separada. Para mas información ver *ITU Recommendation BT.601*.

Este filtro se corresponde con una transformación lineal que puede ser escrita de la siguiente forma:

(a) Imagen Original (b) Filtro FourCombine (d) Filtro ConvertRGBtoYUV

(c) Filtro MaxClose (e) Filtro LinearZoom

Figura 1: Filtros aplicados sobre la imagen Lena

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}14713 & -0{,}28886 & 0{,}436 \\ 0{,}615 & -0{,}51499 & -0{,}10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

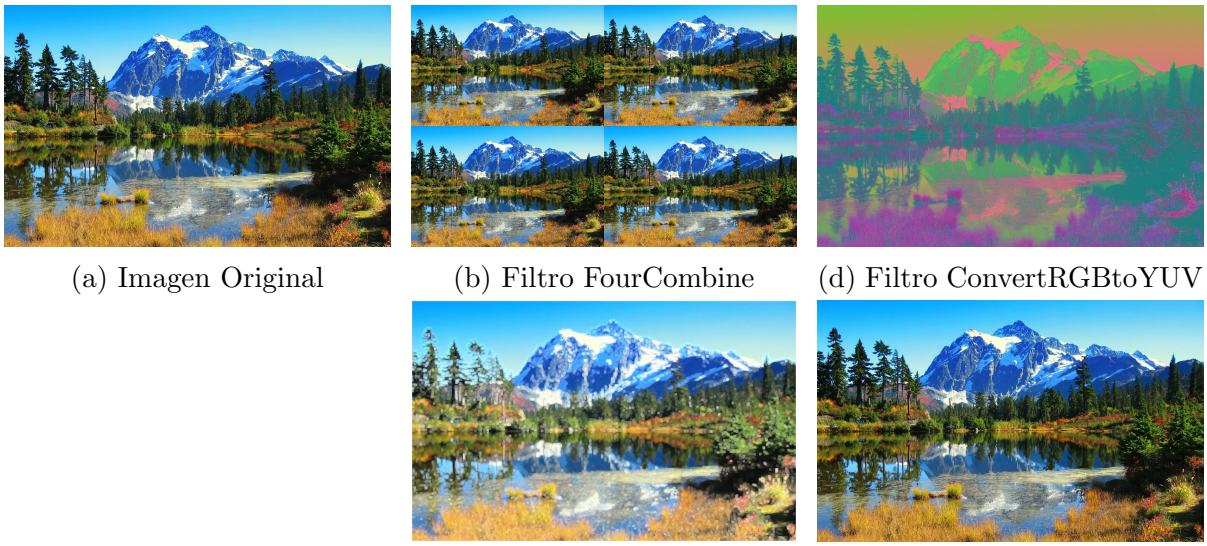
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1{,}13983 \\ 1 & -0{,}39465 & -0{,}58060 \\ 1 & 2{,}03211 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}$$

Donde:

- los valores de `R`, `G` y `B` están en el rango 0 a 1.
- `Y` en el rango 0 a 1
- `U` en el rango $-0,436$ a $0,436$
- `V` en el rango $-0,615$ a $0,615$

Si bien esta conversión es correcta, utilizaremos una modificación de la misma realizada con operaciones sobre enteros.

$$RGBtoYUV(R,G,B) = \begin{bmatrix} Y = satura(((66 \cdot R + 129 \cdot G + 25 \cdot B + 128) >> 8) + 16) \\ U = satura(((-38 \cdot R - 74 \cdot G + 112 \cdot B + 128) >> 8) + 128) \\ V = satura(((112 \cdot R - 94 \cdot G - 18 \cdot B + 128) >> 8) + 128) \end{bmatrix}$$



(a) Imagen Original (b) Filtro FourCombine (d) Filtro ConvertRGBtoYUV

(c) Filtro MaxClose (e) Filtro LinearZoom

Figura 2: Filtros aplicados sobre la imagen Paisaje

$$YUVtoRGB(Y,U,V) = \left[\begin{array}{l} R = sature((298 \cdot (Y-16) + 409 \cdot (V-128) + 128) >> 8) \\ G = sature((298 \cdot (Y-16) - 100 \cdot (U-128) - 208 \cdot (V-128) + 128) >> 8) \\ B = sature((298 \cdot (Y-16) + 516 \cdot (U-128) + 128) >> 8) \end{array} \right]$$

Las aridades de las funciones a implementar son las siguientes:

```
void   C_convertYUVtoRGB(uint8_t* src, uint32_t srcw, uint32_t srch,
                         uint8_t* dst, uint32_t dstw, uint32_t dsth);
void ASM_convertYUVtoRGB(uint8_t* src, uint32_t srcw, uint32_t srch,
                         uint8_t* dst, uint32_t dstw, uint32_t dsth);

void   C_convertRGBtoYUV(uint8_t* src, uint32_t srcw, uint32_t srch,
                         uint8_t* dst, uint32_t dstw, uint32_t dsth);
void ASM_convertRGBtoYUV(uint8_t* src, uint32_t srcw, uint32_t srch,
                         uint8_t* dst, uint32_t dstw, uint32_t dsth);
```

2.2. Combinar

Este filtro consiste en mover los pixeles de una imagen tal que queden ordenados en cuatro cuadrantes según indica el siguiente ejemplo.

Imagen Original

11	12	13	14	15	16	17	18
21	22	23	24	25	26	27	28
31	32	33	34	35	36	37	38
41	42	43	44	45	46	47	48
51	52	53	54	55	56	57	58
61	62	63	64	65	66	67	68
71	72	73	74	75	76	77	78
81	82	83	84	85	86	87	88

Luego de aplicar combinar

11	13	15	17	12	14	16	18
31	33	35	37	32	34	36	38
51	53	55	57	52	54	56	58
71	73	75	77	72	74	76	78
21	23	25	27	22	24	26	28
41	43	45	47	42	44	46	48
61	63	65	67	62	64	66	68
81	83	85	87	82	84	86	88

Considerar que cada valor del ejemplo es un pixel de la imagen.

La aridad de la función a implementar es:

```
void   C_fourCombine(uint8_t* src, uint32_t srcw, uint32_t srch,
                     uint8_t* dst, uint32_t dstw, uint32_t dsth);
void ASM_fourCombine(uint8_t* src, uint32_t srcw, uint32_t srch,
                     uint8_t* dst, uint32_t dstw, uint32_t dsth);
```

2.3. Zoom

El filtro `zoom` aumenta el tamaño de la imagen realizando una interpolación lineal entre pixeles.

Imagen Original

11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44

Luego de aplicar zoom

11	I	12	I	13	I	14	X
I	I	I	I	I	I	I	X
21	I	22	I	23	I	24	X
I	I	I	I	I	I	I	X
31	I	32	I	33	I	34	X
I	I	I	I	I	I	I	X
41	I	42	I	43	I	44	X
X	X	X	X	X	X	X	X

Los valores I corresponden a la interpolación lineal entre dos pixeles lindantes, es decir:

A	$(A+B)/2$	B
$(A+C)/2$	$(A+B+C+D)/4$	$(B+D)/2$
C	$(C+D)/2$	D

Los valores X corresponden a una copia de los extremos, tanto superior como a la izquierda.

La aridad de la función a implementar es:

```
void   C_linearZoom(uint8_t* src, uint32_t srcw, uint32_t srch,
                    uint8_t* dst, uint32_t dstw, uint32_t dsth);
void ASM_linearZoom(uint8_t* src, uint32_t srcw, uint32_t srch,
                    uint8_t* dst, uint32_t dstw, uint32_t dsth);
```

2.4. Maximo cercano

Este filtro calcula la componente de color máxima para cada color sobre un *kernel*[1] dado y genera un pixel. Luego, mezcla este pixel con la imagen original. El *kernel* a utilizar será de 7x7 pixeles.

Primero se calcula el máximo de cada componente para todos los pixeles en el kernel y se forma un nuevo pixel con los máximos por componente.

$$
\begin{aligned}
maxR &= \mathbf{MAX}(\quad \mathsf{src}[i][j][R] \quad | \quad i \in \{-3,...,3\}, \quad j \in \{-3,...,3\} \quad) \\
maxG &= \mathbf{MAX}(\quad \mathsf{src}[i][j][G] \quad | \quad i \in \{-3,...,3\}, \quad j \in \{-3,...,3\} \quad) \\
maxB &= \mathbf{MAX}(\quad \mathsf{src}[i][j][B] \quad | \quad i \in \{-3,...,3\}, \quad j \in \{-3,...,3\} \quad)
\end{aligned}
$$

Luego, se realiza una combinación lineal entre este nuevo pixel y el pixel original.

$$
\begin{aligned}
\mathsf{dst}[i][j][R] &= \mathsf{src}[i][j][R] \cdot val + maxR \cdot (1 - val) \\
\mathsf{dst}[i][j][G] &= \mathsf{src}[i][j][G] \cdot val + maxG \cdot (1 - val) \\
\mathsf{dst}[i][j][B] &= \mathsf{src}[i][j][B] \cdot val + maxB \cdot (1 - val)
\end{aligned}
$$

Para todos los pixeles donde el *kernel* resulte inválido, es decir el margen de 2 pixles de ancho alrededor de toda la imagen, estos se pintarán de blanco.

La aridad de la función a implementar es:

```
void   C_maxClose(uint8_t* src, uint32_t srcw, uint32_t srch,
                  uint8_t* dst, uint32_t dstw, uint32_t dsth, float val);
void ASM_maxClose(uint8_t* src, uint32_t srcw, uint32_t srch,
                  uint8_t* dst, uint32_t dstw, uint32_t dsth, float val);
```

3. Formato BMP

El formato BMP es uno de los formatos de imágenes mas simples: tiene un encabezado y un mapa de bits que representa la información de los pixeles. En este trabajo práctico se utilizará una biblioteca provista por la cátedra para operar con archivos en ese formato. Si bien esta biblioteca no permite operar con archivos con paleta, es posible leer dos tipos de formatos, tanto con o sin transparencia. Ambos formatos corresponden a los tipos de encabezado: BITMAPINFOHEADER (40 bytes) y BITMAPV5HEADER (124 bytes).

El código fuente de la biblioteca está disponible como parte del material, deben seguirlo y entenderlo. Las funciones que deben implementar reciben como entrada un puntero a la imagen. Este puntero corresponde al mapa de bits almacenado en el archivo. El mismo está almacenado de forma particular: **las lineas de la imagen se encuentran almacenadas de forma invertida**. Es decir, en la primera fila de la matriz se encuentra la última línea de la imagen, en la segunda fila se encuentra la anteúltima y así sucesivamente.

[1] `https://en.wikipedia.org/wiki/Kernel_(image_processing)`

3.1. Consideraciones

Tener en cuenta las siguientes características generales,

- El ancho de las imágenes es siempre mayor a 16 píxeles y múltiplo de 4 pixeles.
- No se debe perder precisión en los cálculos (excepto las conversiones).
- Las funciones implementadas en ASM deberán operar con la mayor cantidad posible de bytes.
- El procesamiento de los pixeles se deberá hacer **exclusivamente** con instrucciones **SSE**.
- El trabajo práctico se debe poder ejecutar en las máquinas de los laboratorios.

3.2. Desarrollo

Para facilitar el desarrollo se cuenta con todo lo necesario para poder compilar y probar las funciones a medida que las implementan. Los archivos entregados están organizados de la siguiente forma:

- `src`: Contiene los fuentes del programa principal *tp2*
- `src/bmp`: Contiene los fuentes de la biblioteca de BMP
- `src/filters`: Contiene la implementación de todos los filtros
- `src/tools`: Contiene los fuentes del programa *diff*
- `bin`: Contiene los ejecutables, *tp2* y *diff*
- `img`: Contiene imágenes de prueba
- `test`: Contiene scripts para realizar tests sobre los filtros y uso de la memoria

3.2.1. Compilar

Ejecutar `make` desde la carpeta `src`.

3.2.2. Uso

Ejecutando `./bin/tp2 --help` obtenemos:

```
Uso: ./tp2 <c/asm1/asm2> <fitro> <parametros...>

Opcion C o ASM
         c : ejecuta el codigo C
      asm1 : ejecuta el codigo ASM

Filtro:
        <c/asm> rgb2yuv     <src> <dst>
        <c/asm> yuv2rgb     <src> <dst>
        <c/asm> fourCombine <src> <dst>
        <c/asm> linearZoom  <src> <dst>
        <c/asm> maxCloser   <src> <dst> <val>
```

El programa toma dos parámetros inicialmente y luego una serie de parámetros adicionales dependiendo del filtro seleccionado. En este caso todos los filtros toman el tamaño de la imagen fuente.
El primer parámetro es la implementación a utilizar, puede ser `c` o `asm`. El segundo parámetro es el nombre del filtro, puede ser `rgb2yuv`, `yuv2rgb`, `fourCombine`, `linearZoom` o `maxCloser`. Por ultimo los parámetros adicionales dependiendo de cada filtro y respetando el siguiente:

- `src`: Ruta del archivo de entrada
- `dst`: Ruta del archivo de salida

En Maximo cercano,

- `val`: valor entre 0 y 1 que indica el porcentaje de mezcla

3.2.3. Ejemplo de uso

- `./tp2 asm rgb2yuv ../img/lena.bmp ../img/lena_rgb2yuv_asm.bmp`

Aplica el filtro **rgb2yuv** a la imagen `../img/lena.bmp` utilizando la implementación en `asm` del filtro y almacena el resultado en `../img/lena_rgb2yuv_asm.bmp`

3.2.4. Mediciones de rendimiento

La forma de medir el rendimiento de nuestras implementaciones se realizará por medio de la toma de tiempos de ejecución. Como los tiempos de ejecución son muy pequeños, se utilizará uno de los contadores de performance que posee el procesador.

La instrucción de assembler `rdtsc` permite obtener el valor del Time Stamp Counter (TSC) del procesador. Este registro se incrementa en uno con cada ciclo del procesador. Obteniendo la diferencia entre los contadores antes y despues de la llamada a la función, podemos obtener la cantidad de ciclos de esa ejecución. Esta cantidad de ciclos no es siempre igual entre invocaciones de la función, ya que este registro es global del procesador y se ve afectado por una serie de factores.

Existen principalmente dos problemáticas a solucionar:

1. La ejecución puede ser interrumpida por el *scheduler* para realizar un cambio de contexto, esto implicará contar muchos más ciclos (*outliers*) que si nuestra función se ejecutara sin interrupciones.
2. Los procesadores modernos varian su frecuencia de reloj, por lo que la forma de medir ciclos cambiará dependiendo del estado del procesador.

Para medir tiempos deberán idear e implementar una metodología que les permita evitar estos dos problemas, o al menos reducir su impacto. En el archivo `rdtsc.h` encontrarán las funciones necesarias para implementarla.

3.2.5. Herramientas

En el código provisto, podrán encontrar una herramienta que les permitirá comparar dos imágenes. El código de la misma se encuentra en `src/tools`, y se compila junto con el resto del trabajo práctico. El ejecutable, una vez compilado, se almacenará en `bin/diff`. La aplicación se utiliza desde linea de comandos de la forma:
`./bin/diff <opciones> <archivo_1> <archivo_2> <epsilon>`.

Esto compara los dos archivos según las componentes de cada pixel, siendo epsilon la diferencia máxima permitida entre pixeles correspondientes de las dos imágenes. Tiene dos opciones: listar las diferencias o generar imagenes blanco y negro por cada componente, donde blanco es marca que hay diferencia y negro que no.
Las opciones soportadas por el programa son:

-i, --image	Genera imagenes de diferencias por cada componente
-v, --verbose	Lista las diferencias de cada componente y su posición en la imagen
-a, --value	Genera las imagenes mostrando el valor de la diferencia
-s, --summary	Muestra un resumen de diferencias

3.2.6. Tests

Para verificar el correcto funcionamiento de los filtros sin tener que hacer pruebas manualmente. Se provee un conjunto de scripts que se encuentran en la carpeta `solucion/tests`. Para utilizarlos se deben realizar los siguientes pasos:

1- Generar las imágenes para probar.
Ejecutar el script `./1_generar_imagenes.sh`

2- Correr los casos con el TP implementado por la cátedra y generar los resultados de cada filtro.
Ejecutar el script `./2_generar_resultados_catedra.sh`

3- Correr los casos con el TP implementado por ustedes. Genera resultados y chequea **diferencias** con los resultados de la cátedra.
Ejecutar el script `./3_correr_tests_diff.sh`

4- Correr los casos con el TP implementado por ustedes. Genera resultados y chequea **uso de memoria**.
Ejecutar el script `./4_correr_tests_mem.sh`

Una vez ejecutados los dos primeros pasos, es posible ejecutar los siguientes sin tener que ejecutar los primeros nuevamente.

3.3. Informe

El informe debe incluir las siguientes secciones:

1. Carátula: La carátula del informe con el **número/nombre del grupo**, los **nombres y apellidos** de cada uno de los integrantes junto con **número de libreta** y **email**.

2. Introducción: Describe lo realizado en el trabajo práctico. (y si quedó algo sin realizar)

3. Desarrollo: Describe **en profundidad** cada una de las funciones que implementaron. Para la decripción de cada función deberan decir cómo opera una iteración del ciclo de la función. Es decir, cómo mueven los datos de la imagen a los registros, cómo los reordenan para procesarlos, las operaciones que se aplican a los datos, etc. Para esto pueden utilizar pseudocódigo, diagramas (mostrando gráficamente el contenido de los registros **XMM**) o cualquier otro recurso que les resulte útil para describir la adaptación del algoritmo al procesamiento vectorial. No se deberá incluir el código assembler de las funciones (aunque se pueden incluir extractos en donde haga falta).

4. Resultados: **Deberán analizar** y **comparar** las implementaciones de las funciones en su versión **C** y **assembler** y mostrar los resultados obtenidos a través de tablas y gráficos. Para esto deberán plantear experimentos que les permitan medir el rendimiento y comparar entre las implementaciones. Deberán además explicar detalladamente los resultados obtenidos y analizarlos. En el caso de encontrar anomalías o comportamientos no esperados deberán construir nuevos experimentos para entender qué es lo que sucede.

5. Conclusión: Reflexión final sobre los alcances del trabajo práctico, la programación vectorial a bajo nivel, problemáticas encontradas, y todo lo que consideren pertinente.

El informe no puede exceder las **20** páginas, sin contar la carátula.

Importante: El informe se evalúa de manera independiente del código. Puede reprobarse el informe y en tal caso deberá ser reentregado para aprobar el trabajo práctico.

4. Entrega

Se deberá entregar el solamente contenido de la carpeta `src` junto con el informe.

La fecha de entrega de este trabajo es **09/05**. Deberá ser entregado a través de un repositorio GIT almacenado en `https://git.exactas.uba.ar` respetando el protocolo enviado por mail y publicado en la página web de la materia.

Ante cualquier problema con la entrega, comunicarse por mail a la lista de docentes `orga2-doc@dc.uba.ar`.