Trabajo práctico 2

Organización del Computador II

Primer cuatrimestre 2012

1. Introducción

El objetivo de este trabajo práctico es explorar el modelo de programación SIMD. Una aplicación popular del modelo SIMD es el procesamiento de imágenes. En este trabajo práctico se implementarán filtros para procesamiento de imágenes, utilizando lenguaje ensamblador e instrucciones SIMD. Los filtros a implementar se describen a continuación.

1.1. Rotar

Consiste en rotar los canales de color entre si, de la siguiente manera:

$$\begin{aligned} \mathbf{R} &\longrightarrow \mathbf{G} \\ \mathbf{G} &\longrightarrow \mathbf{B} \\ \mathbf{B} &\longrightarrow \mathbf{R} \end{aligned}$$

1.2. Smalltiles

Esta operación consiste en repetir la imagen original 4 veces, de forma más chica en la imagen destino. Es decir, que si se tiene en destino una imagen de tamaño w x h, en el destino se tendrán 4 imágenes de tamaño $w/2$ x $h/2$, una en cada cuadrante.

Para copiar los píxeles, a cada (i, j) de la primera imagen destino, le corresponde el valor de la posición $(2i, 2j)$ en la imagen fuente.

Nota: En el caso que la entrada sea una imagen impar se debe duplicar la linea que sobra en su extremo derecho e inferior según corresponda.

1.3. Blit

Esta operación recibe más parámetros que las demás; además de tener las imágenes fuente y destino (de tamaño w x h), se recibe una imagen adicional **blit** también con su respectivo tamaño (bw x bh), que se utilizará a modo de máscara. Se cumple que $bw \leq w \wedge bh \leq h$.

La aplicación de este filtro consiste en generar una nueva imagen combinando la original con el **blit**. Un color de la imagen **blit** es considerado como transparente, para que en la combinación algunos pixeles del **blit** queden por encima de la imagen original y otros no se vean.

Para este trabajo práctico, la imagen **blit** será una imagen de Perón. Decimos que una imágen ha sido "Peronizada" cuando en su extremo inferior derecho podemos ver a Perón agitando su brazo.

Para cada píxel p en la imagen de Perón (blit):

$$dst(p) = \begin{cases} src(p) & \text{si } blit(p) \text{ es de color magenta, es decir sus colores son } (255, 0, 255) \\ blit(p) & \text{si no} \end{cases}$$

Es decir que si el píxel en la imagen de Perón es magenta, entonces el nuevo píxel tendrá los colores del píxel correspondiente en la imagen original; y si no los colores del píxel en la imagen de Perón.

Las columnas y filas que no se puedan procesar de esta manera, es decir, las primeras $h - bh$ filas; y de las filas restantes, las primeras $w - bw$ columnas; quedarán igual que como estaban originalmente.

Nota: como debe valer la restricción $bw \leq w \wedge bh \leq h$, esta operación no podrá ser aplicada a imágenes cuyo tamaño sea menor a 89 píxeles de ancho x 128 de alto.

1.4. Monocromatizar

Para convertir una imagen a escala de grises debemos contar con una función que sea capaz de monocromatizar una imagen a color. La función más sencilla para hacer esto es:

$$f(p) = \alpha R + \beta G + \gamma B$$

La función f se aplica a cada píxel p de la imagen; R, G, B son sus componentes de color y α, β y γ son los coeficientes 0.21, 0.71 y 0.07 respectivamente.
El resultado $f(p)$ debe replicarse en las 3 componentes del píxel para lograr el efecto:

$$\begin{aligned} \mathbf{R} &\longrightarrow f(p) \\ \mathbf{G} &\longrightarrow f(p) \\ \mathbf{B} &\longrightarrow f(p) \end{aligned}$$

1.5. Sepia

Esta operación consiste en cambiar la información de color de cada píxel de la siguiente manera:

$$\begin{aligned} \mathbf{R} &\longrightarrow \text{tmp} * 0.5 \\ \mathbf{G} &\longrightarrow \text{tmp} * 0.3 \\ \mathbf{B} &\longrightarrow \text{tmp} * 0.2 \end{aligned}$$

donde $tmp = R + G + B$.

1.6. Edge

Puede definirse como un borde a los píxeles donde la intensidad de la imagen cambia de forma abrupta. Si se considera una función de intensidad de la imagen, entonces lo que se busca son saltos en dicha función. La idea básica detrás de cualquier detector de bordes es el cálculo

de un operador local de derivación.

Vamos a usar en este caso el operador de Laplace, cuya matriz es

$$M = \begin{pmatrix} 0{,}5 & 1 & 0{,}5 \\ 1 & -6 & 1 \\ 0{,}5 & 1 & 0{,}5 \end{pmatrix}$$

Para obtener los bordes, se posiciona el centro de la matríz de Laplace en cada posición (x, y) y se realiza la siguiente operación

$$dst(x,y) = \sum_{k=0}^{2} \sum_{l=0}^{2} src(x+k-1, y+l-1) * M(k,l)$$

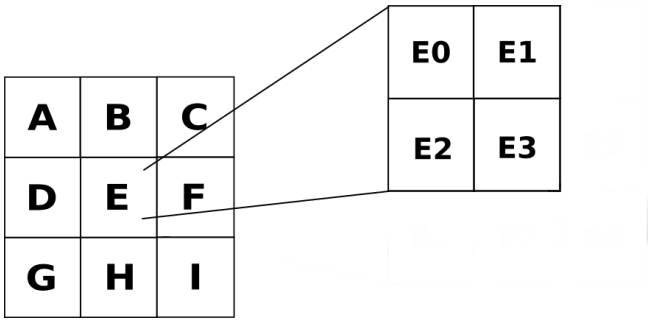
Es decir $dst(x, y) =$

$$\begin{array}{ccccccc} src(x-1,y-1)*M_{00} & + & src(x-1,y)*M_{01} & + & src(x-1,y+1)*M_{02} & + \\ src(x,y-1)*M_{10} & + & src(x,y1)*M_{11} & + & src(x,y+1)*M_{12} & + \\ src(x+1,y-1)*M_{20} & + & src(x+1,y)*M_{21} & + & src(x+1,y+1)*M_{22} & \end{array}$$

Este filtro opera sobre imágenes en escala de grises (1 componente de color por píxel)

1.7. Scale 2x

El escalado de una imagen consiste en expandir un píxel combinando sus datos con los datos de sus vecinos.



La combinación respeta las siguientes reglas:

- Si $B \neq H$ y $D \neq F$
  - Si $D = B \Rightarrow E_0 = D$, si no $E_0 = E$
  - Si $B = F \Rightarrow E_1 = F$, si no $E_1 = E$
  - Si $D = H \Rightarrow E_2 = D$, si no $E_2 = E$
  - Si $H = F \Rightarrow E_3 = F$, si no $E_3 = E$
- si no
  - $E_0 = E_1 = E_2 = E_3 = E$

La operación se realiza solamente sobre el primer cuarto de la imagen, es decir tomando la primera mitad de las filas y de las columnas. Deberá procesarse a partir del pixel en la posición (1,1).

Para mas información: `http://scale2x.sourceforge.net/algorithm.html`

1.8. Blur

Para lograr un efecto de suavizado en una imágen, se realiza un promedio del valor de cada píxel con el de sus vecinos. En este caso para cada posición, se tienen en cuenta los vecinos cuya distancia es ≤ 2.

Es decir $blur(x, y) =$

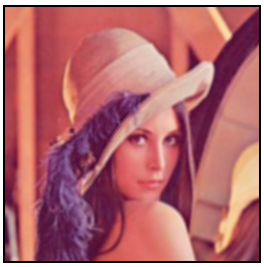
$$(src(x, y-2) + src(x-1, y-1) + src(x, y-1) + src(x+1, y-1) +$$
$$src(x, y-2) + src(x, y-1) + src(x, y) + src(x, y+1) + src(x, y+2) +$$
$$src(x-1, y+1) + src(x, y+1) + src(x+1, y+1) + src(x, y+2))/13$$

1.9. Ejemplos

Los resultados de aplicar los filtros son los siguientes:


Imagen original

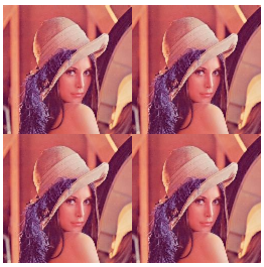

Monocromatizar


Blur

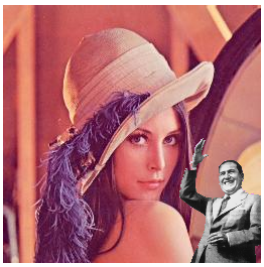

Scale 2x


Edge


Sepia


Smalltiles


Rotar


Blit

2. Enunciado

Uno de los objetivos de este trabajo práctico es analizar la performance de un procesador al hacer uso de las operaciones **SIMD** para el procesamiento de imágenes.

Se implementarán ocho funciones, cada una en dos versiones: una en lenguaje C, y una en lenguaje ensamblador haciendo uso de las instrucciones **SSE**.

Finalmente, se compararán ambas versiones analizando las mejoras de performance obtenidas.

2.1. Código

Implementar los filtros descriptos anteriormente, tanto en lenguaje C como en lenguaje ensamblador. Más precisamente, deberán implementar las siguientes funciones:

- *void rotar_asm(unsigned char *src, unsigned char *dst, int h, int w, int src_row_size, int dst_row_size)*
- *void rotar_c(unsigned char *src, unsigned char *dst, int h, int w, int src_row_size, int dst_row_size)*
- *void smalltiles_asm(unsigned char *src, unsigned char *dst, int h, int w, int src_row_size, int dst_row_size)*
- *void smalltiles_c(unsigned char *src, unsigned char *dst, int h, int w, int src_row_size, int dst_row_size)*
- *void blit_c (unsigned char *src, unsigned char *dst, int h, int w, int src_row_size, int dst_row_size, unsigned char *blit, int bh, int bw, int b_row_size)*
- *void blit_v (unsigned char *src, unsigned char *dst, int h, int w, int src_row_size, int dst_row_size, unsigned char *blit, int bh, int bw, int b_row_size)*
- *void monocromatizar_c (unsigned char* src, unsigned char* dst, int h, int w, int src_row_size, int dst_row_size)*
- *void monocromatizar_asm (unsigned char* src, unsigned char* dst, int h, int w, int src_row_size, int dst_row_size)*
- *void sepia_asm(unsigned char *src, unsigned char *dst, int h, int w, int src_row_size, int dst_row_size)*
- *void sepia_c(unsigned char *src, unsigned char *dst, int h, int w, int src_row_size, int dst_row_size)*
- *void edge_asm(unsigned char *src, unsigned char *dst, int h, int w, int src_row_size, int dst_row_size)*
- *void edge_c(unsigned char *src, unsigned char *dst, int h, int w, int src_row_size, int dst_row_size)*
- *void scale2x_asm(unsigned char *src, unsigned char *dst, int h, int w, int src_row_size, int dst_row_size)*
- *void scale2x_c(unsigned char *src, unsigned char *dst, int h, int w, int src_row_size, int dst_row_size)*
- *void blur_asm(unsigned char *src, unsigned char *dst, int h, int w, int src_row_size, int dst_row_size)*
- *void blur_c(unsigned char *src, unsigned char *dst, int h, int w, int src_row_size, int dst_row_size)*

Nota: Sólo *edge* funciona con imágenes en escala de grises.

- *src*: Es el puntero al inicio de la matriz de elementos de 32 bits sin signo que corresponde a la imagen de entrada. El primer byte es al canal azul de la imagen (B), el segundo el verde (G) y el tercero el rojo (R). El cuarto canal puede considerarse "de alineación". Puede asumirse que el valor de este canal es siempre 0. En el caso de una imagen en escala de grises, sólo tiene un canal de 8 bits sin signo.
- *dst*: Es el puntero al inicio de la matriz de elementos de 32 bits sin signo que representa a la imagen de salida. Puede asumirse que el valor del cuarto canal ("de alineación") es desestimado al momento de guardar la imagen. En el caso de una imagen en escala de grises, cada elemento es de 8 bits sin signo.
- *h*: Representa el alto en píxeles de la imagen, es decir, la cantidad de filas de las matrices de entrada y salida.

- *w*: Representa el ancho en píxeles de la imagen, es decir, la cantidad de columnas de las matrices de entrada y salida.

- *src_row_size*: Representa la cantidad de bytes que ocupa una fila de la matriz de entrada. Dependiendo del formato se extiende el tamaño (en bytes) de las filas de la imagen, de forma que sea múltiplo de un valor conveniente para su posterior acceso, por ejemplo, 16. Esto no afecta a la imagen, pero se debe tener en cuenta a la hora de recorrer la misma. Por ejemplo, si una imagen tiene 10 píxeles de ancho ocuparía 10 bytes, si se la extiende a 12 bytes, corresponderán dos bytes de relleno al final de cada linea. No debe accederse a estos bytes de relleno.

- *dst_row_size*: Idem *src_row_size* pero para la imagen destino.

2.1.1. Consideraciones

Las funciones a implementar en lenguaje ensamblador deben utilizar el set de instrucciones **SSE**, a fin de optimizar la performance de las mismas.

Tener en cuenta lo siguiente:

- El ancho de las imágenes es siempre mayor a 16 píxeles.
- No se debe perder precisión en ninguno de los cálculos.
- La implementación de cada filtro deberá estar optimizada para el filtro que se está implementando. No se puede hacer una función que aplique un filtro genérico y después usarla para implementar los que se piden.
- Para el caso de las funciones implementadas en lenguaje ensamblador, deberán trabajar con **al menos 8 bytes simultámeamente**, procesando la mayor cantidad de píxeles posibles según el caso.
- Para los filtros en los que no se puedan procesar los bordes, estos quedarán en negro.
- En el caso de filtros donde se vea afectado el tamaño original de la imágen (*smalltiles*, *scale2x*) para imagenes de ancho impar habrá una tolerancia de 1 píxel en los bordes de la imágen. Deberán justificar en el caso de que esto suceda.
- Puede asumir que la cuarta componente de color es siempre 0 y no se tiene en cuenta al final.
- Puede asumir que la imagen de destino se inicializa en negro.
- El procesamiento de los píxeles se deberá hacer **exclusivamente** con instrucciones **SSE**, no está permitido procesarlos con registros de propósito general.
- El TP se tiene que poder ejecutar en las máquinas del laboratorio.

2.2. Desarrollo

Para facilitar el desarrollo del trabajo práctico se cuenta con todo lo necesario para poder compilar y probar las funciones que vayan a implementar. Dentro de los archivos presentados deben completar el código de las funciones pedidas. Puntualmente encontrarán el programa principal (de línea de comandos), denominado **tpcopados**, que se ocupa de parsear las opciones ingresadas por el usuario y ejecutar el filtro seleccionado sobre la imagen ingresada.

Para la manipulación de las imágenes (cargar, grabar, etc.) el programa hace uso de la biblioteca **OpenCV**, por lo que no se requiere implementar estas funcionalidades. Para instalar esta biblioteca en las distribuciones basadas en **Debian** basta con ejecutar:

$ sudo apt-get install libcv-dev libhighgui-dev libcvaux-dev python-bottle python-dev

Los archivos entregados están organizados en las siguientes carpetas:

- *bin*: Contiene el ejecutable del TP
- *imgs*: Contiene imágenes de prueba
- *data*: Contiene imágenes necesarias para el procesamiento
- *enunciado*: Contiene este enunciado
- *solucion*: Contiene los fuentes del programa principal, junto con su respectivo **Makefile** que permite compilar el programa.
- *tester*: Contiene un visualizador de imágenes para verificar los filtros.

El uso del programa principal es el siguiente:

$ ./tpcopados <opciones> <nombre_filtro> <nombre_archivo_entrada> [parámetros]

Soporta los tipos de imágenes más comunes y acepta las siguientes opciones:

- *nombre_filtro*
  Los filtros que se pueden aplicar son:
  - monocromatizar
  - blur
  - scale2x
  - sepia
  - smalltiles
  - rotar
  - blit
  - edge
- *-h, –help*
  Imprime la ayuda
- *-i, –implementacion NOMBRE_MODO*
  Implementación sobre la que se ejecutará el proceso seleccionado. Los implementaciones disponibles son: c, asm
- *-t, –tiempo CANT_ITERACIONES*
  Mide el tiempo que tarda en ejecutar el filtro sobre la imagen de entrada una cantidad de veces igual a CANT_ITERACIONES

- *-v, –verbose*
  Imprime información adicional

Por ejemplo:

$ ./tpcopados -i asm blur ../imgs/tests/lena1024x1024.jpg

Aplica el filtro **blur** a la imagen seleccionada utilizando la implementación en lenguaje ensamblador del filtro.

Si hacemos:

$ ./tpcopados -t 1000 -i asm blur ../imgs/tests/lena1024x1024.jpg

Realiza el mismo proceso anterior pero repite la aplicación del filtro dentro de un ciclo de **1000** iteraciones y devuelve la cantidad de **ticks** (ciclos de reloj del procesador) que insumió la aplicación del filtro. Esto será utilizado para comparar la performance de las versiones en C y assembler.

Nota: Para evitar arrastrar errores, la aplicación de los filtros no utiliza las funciones de conversión a escala de gris implementada por ustedes sino que utiliza la que provee la biblioteca **OpenCV**.
Importante nota: el tester verifica el contenido de la carpeta *imgs/tests*. Si se procesa sobre esa carpeta, se crearan nuevos archivos. Se recomienda **fuertemente** no trabajar sobre esa carpeta.

2.2.1. Mediciones de tiempo

Utilizando la instrucción de assembler `rdtsc` podemos obtener el valor del Time Stamp Counter (TSC) del procesador. Este registro se incrementa en uno con cada ciclo del procesador. Restando el valor del registro antes de llamar a una función al valor del registro luego de la llamada, podemos obtener la duración en ciclos de esa ejecución de la función.

Esta cantidad de ciclos no es siempre igual entre invocaciones de la función, ya que este registro es global del procesador y si nuestro programa es interrumpido por el scheduler para realizar un cambio de contexto, contaremos muchos más ciclos que si nuestra función se ejecutara sin interrupción. Por esta razón el programa principal del TP permite especificar una cantidad de iteraciones para repetir el filtro, con el objetivo de suavizar este tipo de outliers.

A modo de ejemplo se presentan algunos resultados obtenidos por la cátedra para implementaciones en ensamblador. Este experimento fue corrido en la PC X del laboratorio 4 para la imagen *bosque4000x3000.jpg*.

Filtro	Ticks (x10^6)
rotar	686.481
smalltiles	613.392
blit	673.432
monocromatizar	834.353
sepia	881.647
edge	475.658
scale2x	588.749
blur	3836.1

Cuadro 1: Tiempo de los experimentos realizados por la cátedra

2.2.2. Verificación

Para verificar el correcto funcionamiento de los filtros, se proveen la herramienta **tester**:

Es una aplicación que permite visualizar el resultado de aplicar un filtro a una imágen, el resultado de comparar la imagen filtrada en C y en ensamblador con la correctamente filtrara por la cátedra y datos de los tiempos obtenidos.

Esta herramienta se encuentra en la carpeta *tester*. Se ejecuta de la siguiente forma:

$ python webserv.py

Una vez ejecutada deberá abrir un navegador web (preferiblemente google chrome aunque también está verificado para firefox) y abrir la dirección `http://localhost:8081`

Las imágenes que da como opciones para procesar se encuentran en la carpeta *tester/img*. Cada vez que se compile el código fuente del ejecutable *tpcopados* usando make se reiniciará el tester automáticamente en el lapso de 2 segundos con los cambios realizados en el código. En el caso de que el filtro ejecutado falle y no finalice la ejecución correctamente, el tester se cerrará.

Esta herramienta verifica el correcto procesamiento de las imágenes con las realizadas por la cátedra. Realiza una comparación píxel a píxel y devuelve las siguientes métricas:

- Píxeles procesados: cantidad de píxeles que se compararon.
- Píxeles distintos: cantidad de píxeles cuyos valores no coincidieron.
- Porcentaje: relación entre cantidad de píxeles procesados y distintos. Valor entre 0 y 1.
- Diferencia máxima: es el máximo valor que se obtuvo entre dos píxeles distintos.
- Diferencia acumulada: es la suma de las diferencias entre los píxeles distintos.
- Radio: es la relación entre la diferencia acumulada y la cantidad de píxeles procesados.

Esta herramienta no verifica el correcto uso de la memoria.

Deberá verificar el correcto uso de la memoria utilizando *valgrind*. Se recomienda realizarlo para todas las implementaciones de todos los filtros en todas las imágenes provistas por la cátedra.

3. Informe

El informe debe incluir las siguientes secciones:

a) 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**.

b) Introducción: Describe lo realizado en el trabajo práctico.

c) Desarrollo: Describe **en profundidad** cada una de las funciones que implementaron. Para la decripción de cada función deberán decir cómo opera una iteración del ciclo de la función. Es decir, cómo mueven los datos de la imagen a los registros, cómo los procesan, 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 le sea útil para describir la adaptación del algoritmo a procesamiento vectorial. No se deberá incluir el código assembler de las funciones (aunque se pueden incluir extractos en donde haga falta). Debe utilizar la herramienta `tester` para verificar el correcto procesamiento. En el caso de obtener

diferencias deberan justificarlas fuertemente describiendo porque no puede eliminarse dicha diferencia. Es motivo de reentrega el tener diferencias con los filtros aplicados por la cátedra sin justificación coherente que lo avale.

d) Resultados: **Deberán analizar** y **comparar** las implementaciones de cada funciones en su versión **C** y **assembler** y mostrar los resultados obtenidos a través de tablas y gráficos. También deberán comentar sobre los resultados que obtuvieron. En el caso de que sucediera que la versión en C anduviese más rápido que su versión en assembler **justificar fuertemente** a qué se debe esto. Es requisito para la aprobación realizar un análisis exahustivo de experimentos y tiempos y presentar los datos en forma de gráficos. El uso de tablas con valores se aceptará sólo a modo de apéndice (en una sección con dicho nombre) para justificar los gráficos y demás herramientas utilizadas para el análisis.

e) 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 ni el apéndice.

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 un archivo comprimido sólo con dos carpetas:

- `informe`: Versión digital del informe entregado.
- `src`: la carpeta *src* que fue dado para realizarlo, habiendo modificado sólo los archivos que tienen como nombre las funciones a implementar.

Adicionalmente, deberá concurrir a clase el día de la entrega con una versión impresa del informe entregado digitalmente.

La fecha de entrega de este trabajo es **martes 8 de mayo** y deberá ser entregado a través de la página web. El sistema sólo aceptará entregas de trabajos hasta las **16:59** del día de entrega.

Ante cualquier problema con la entrega, comunicarse por mail a la lista de docentes.