Trabajo práctico 2

Organización del Computador II

Segundo cuatrimestre 2011

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. 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:

$$f(p) = \sqrt[\epsilon]{\alpha R^\epsilon + \beta G^\epsilon + \gamma B^\epsilon}$$

La función f se aplica a cada píxel p de la imagen; R, G, B son sus componentes de color, α, β y γ son coeficientes entre 0 y 1, y ϵ es un exponente entre 1 e ∞.

En nuestro caso se implementará la función de conversión a escala de grises para los casos extremos, donde ϵ en la función vale $\epsilon = 1$ o $\epsilon = \infty$. Además, se fijarán los parámetros restantes en $\alpha = 1/4$, $\beta = 1/2$ y $\gamma = 1/4$. Luego, tenemos dos posibles funciones para monocromatizar:

$$I_{out\_uno}(p) = \frac{(R+2G+B)}{4}(\epsilon = 1)$$

$$I_{out\_infinito}(p) = max(R, G, B)(\epsilon = \infty)$$

(Cada píxel p se considera compuesto por tres componentes R, G y B.)

1.2. Separar canales

Esta función se encarga de separar la imagen por canales, en Rojo(R), Verde(G) y Azul(B); en tres imágenes distintas.

$$I_{out\_red}(p) = R$$
$$I_{out\_green}(p) = G$$
$$I_{out\_blue}(p) = B$$

1.3. Umbral (thresholding)

Genera una imagen de tres colores, blanco, gris y negro, determinada por la imagen fuente respetando la siguiente función:

$$I_{out}(p) = \begin{cases} 0 & p \leq umbral\_minimo \\ 128 & umbral\_minimo < p \leq umbral\_maximo \\ 255 & p > umbral\_maximo \end{cases}$$

1.4. Invertir

Genera una imagen donde el rango de la escala de grises de la imagen esta invertido. Para esto aplica la siguiente función:

$$I_{out}(p) = MAX - p$$

Donde MAX corresponde al máximo valor posible para un píxel de la imagen.

1.5. Suavizado Gaussiano (Gaussian smoothing)

Este filtro corresponde a aplicar un efecto de suavizado sobre una imagen por medio de una función Gaussiana. El resultado visual es una reducción del ruido y una perdida de detalle en la imagen, equivalente a un efecto "fuera de foco".

La aplicación de este filtro se realiza utilizando máscaras. Éstas corresponden a la aplicación en el plano de la discretización de la función de Gauss.

En nuestro caso para matrices de 3×3 se construye de la siguiente forma:

$$\tfrac{1}{4} \cdot \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot \tfrac{1}{4} \cdot \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

La matriz o operador resultante corresponde a:

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

Concretamente la aplicación de este filtro se resume en el siguiente cálculo:

$$\begin{aligned} I_{out}(i,j) = \; & I_{in}(i-1, j-1) \cdot 1/16 + I_{in}(i-1, j) \cdot 2/16 + I_{in}(i-1, j+1) \cdot 1/16 + \\ & I_{in}(i+0, j-1) \cdot 2/16 + I_{in}(i+0, j) \cdot 4/16 + I_{in}(i+0, j+1) \cdot 2/16 + \\ & I_{in}(i+1, j-1) \cdot 1/16 + I_{in}(i+1, j) \cdot 2/16 + I_{in}(i+1, j+1) \cdot 1/16 \end{aligned}$$

Donde I_{in} es la imagen fuente y I_{out} el destino. Los indices i y j corresponden a las coordenadas en la imagen.

Para generar el píxel $I_{out}(i, j)$ en la imagen destino, se debe operar sobre enteros, por lo que la aplicación de la función descripta debe ser adaptada para tal caso.

1.6. Normalizar (contrast stretching)

La normalización es el proceso que modifica el rango de intensidad de los píxeles en una imagen. El objetivo es convertir una imagen a un deteminado rango en escala de grises. Por ejemplo, si el rango de intensidad en una imagen oscila entre 20 y 100, la normalización modifica su rango por otro, como por ejemplo de 0 a 255. En nuestro caso vamos a operar con el rango máximo de la imagen. La función utilizada para realizar este cálculo es la siguiente:

$$I_{out}(i,j) = (I_{in}(i,j) - c) \cdot \left(\frac{a-b}{c-d}\right) + a$$

Donde:

- a : Máximo valor posible
- b : Mínimo valor posible
- c : Máximo valor en la imagen
- d : Mínimo valor en la imagen

1.7. Ejemplos

Los resultados de pasar a escala de grises y aplicar los filtros son los siguientes:

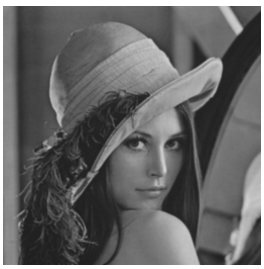
Imagen original

Monocromatizar $\epsilon = 1$

Monocromatizar $\epsilon = \infty$

Umbralizar (min 64, max 128)

Invertir



Suavizar

Normalizar

Canal Rojo

Canal Verde

Canal Azul

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 seis 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 para imágenes en color (24 bits):

- *void monocromatizar_uno_c (unsigned char* src, unsigned char* dst, int h, int w, int src_row_size, int dst_row_size)*
- *void monocromatizar_inf_c (unsigned char* src, unsigned char* dst, int h, int w, int src_row_size, int dst_row_size)*
- *void separar_canales_c (unsigned char* src, unsigned char* dst_r, unsigned char* dst_g, unsigned char* dst_b, int h, int w, int src_row_size, int dst_row_size)*
- *void monocromatizar_uno_asm (unsigned char* src, unsigned char* dst, int h, int w, int src_row_size, int dst_row_size)*
- *void monocromatizar_inf_asm (unsigned char* src, unsigned char* dst, int h, int w, int src_row_size, int dst_row_size)*
- *void separar_canales_asm (unsigned char* src, unsigned char* dst_r, unsigned char* dst_g, unsigned char* dst_b, int h, int w, int src_row_size, int dst_row_size)*

- *src*: Es el puntero al inicio de la matriz de elementos de 24 bits sin signo (el primer byte corresponde al canal azul de la imagen (B), el segundo el verde (G) y el tercero el rojo (R)) que representa a la imagen de entrada. Es decir, como la imagen está en color, cada píxel está compuesto por 3 bytes.
- *dst*: Es el puntero al inicio de la matriz de elementos de 8 bits sin signo que representa a la imagen de salida. Como la imagen está en escala de grises cada píxel se corresponde con un elemento de la matriz. Tanto *dst_r*, *dst_g* y *dst_b* indican el puntero al inicio de la matriz de salida para cada uno de los canales.
- *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 tama no (en bytes) de las filas de la imagen, de forma que sea múltiplo de un valor conveniente para su posterior acceso, por ejemplo, 4. 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 pixeles de ancho ocuparía 10 bytes, si se la extiende a 12 bytes, corresponderan dos bytes de basura al final de cada linea.

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

Para imágenes en escala de grises:

- *void umbral_c (unsigned char* src, unsigned char* dst, int h, int w, int row_size, unsigned char umbral_min, unsigned char umbral_max)*
- *void invertir_c (unsigned char* src, unsigned char* dst, int h, int w, int row_size)*
- *void suavizar_c (unsigned char* src, unsigned char* dst, int h, int w, int row_size)*
- *void normalizar_c (unsigned char* src, unsigned char* dst, int h, int w, int row_size)*

- *void umbral_asm (unsigned char* src, unsigned char* dst, int h, int w, int row_size, unsigned char umbral_min, unsigned char umbral_max)*
- *void invertir_asm (unsigned char* src, unsigned char* dst, int h, int w, int row_size)*
- *void suavizar_asm (unsigned char* src, unsigned char* dst, int h, int w, int row_size)*
- *void normalizar_asm (unsigned char* src, unsigned char* dst, int h, int w, int row_size)*

Donde se respetan los mismos parámetros que en el caso anterior a diferencia de:

- Las imágenes están en escala de grises por lo que cada píxel se corresponde con un elemento de la matriz.
- *umbral_min*: Corresponde a un número entero de umbral mínimo para la comparación en el correspondiente filtro.
- *umbral_max*: Corresponde a un número entero de umbral máximo para la comparación en el correspondiente filtro.
- *row_size*: Idem *src_row_size* pero para ambas imágenes, tanto fuente como 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 imagenes 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 el filtro de Suavizado Gaussiano no se deben procesar los bordes.
- El procesamiento de los píxeles se deberá hacer **exclusivamente** con instrucciones **SSE**, no está permitido procesarlos con registros de proposito 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 **tp2**, 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 imagenes (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
```

Los archivos entregados estan organizados en las siguientes carpetas:

- *bin*: Contiene el ejecutable del TP
- *data*: Contiene imágenes de prueba
- *enunciado*: Contiene este enunciado
- *src*: Contiene los fuentes del programa principal, junto con su respectivo **Makefile** que permite compilar el programa.
- *test*: Contiene scripts para realizar tests sobre los filtros y uso de la memoria.

El uso del programa principal es el siguiente:

```
$ ./tp2 <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:
  - invertir
  - monocromatizar_inf

- monocromatizar_uno
- normalizar
- separar_canales
- suavizar
- umbralizar
  Parámetros : umbral_minino [0, 255], umbral_maximo [0, 255] (umbral_minino $\leq$ umbral_maximo)
  Ejemplo de uso : ./tp2 -i c umbralizar lena.bmp 65 140

- *-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:

```
$ ./tp2 -i asm suavizar ../data/lena.bmp
```

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

Si hacemos:

```
$ ./tp2 -t 1000 -i asm suavizar ../data/lena.bmp
```

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 al 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**.

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.

2.3. 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 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 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).

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.

e) Conclusión: Reflexion final sobre los alcances del trabajo práctico, la programación vectorial a bajo nivel, problematicas 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.

3. Entrega

Se deberá entregar un archivo comprimido con el mismo contenido que el dado para realizarlo, habiendo modificado sólo los archivos que tienen como nombre las funciones a implementar.

La fecha de entrega de este trabajo es **martes 4 de octubre** y deberá ser entregado a través de la página web. El sistema sólo aceptará entregas de trabajos hasta las **23:59** del día de entrega.

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