

Trabajo Práctico 2

Organización del Computador II

Segundo Cuatrimestre 2015

1. Introducción

En este trabajo práctico buscamos una primera aproximación al modelo de procesamiento SIMD (*Single Instruction Multiple Data*). Con este objetivo, el trabajo práctico se compone de dos partes igualmente importantes. En primera instancia aplicaremos lo aprendido en clase programando de manera vectorizada; luego haremos un análisis experimental de los rendimientos obtenidos.

Como campo de aplicación tomamos el procesamiento de imágenes. Deberán implementar dos filtros: blur Gaussiano y diferencia de imágenes, cada uno de ellos en C y en lenguaje ensamblador, para luego plantear hipótesis, experimentar y sacar conclusiones respecto de cada implementación.

Esto último debe llevarse a cabo con un carácter científico y con las metodologías correspondientes. Para facilitarles esta tarea cada grupo tendrá asignado un docente que los guiará en la rigurosidad y exhaustividad del análisis que realicen. Además dedicaremos una clase práctica a estos temas.

1.1. Preliminares

Consideramos a una imagen como una matriz de píxeles. Cada píxel está determinado por cuatro componentes: los colores azul (**b**), verde (**g**) y rojo (**r**), y la transparencia (**a**). En nuestro caso particular estas componentes tendrán 8 bits (1 byte) de profundidad, es decir que estarán representadas por números enteros en el rango $[0, 256)$.

Dada una imagen I , notaremos $I_{x,y}^k$ al valor de la componente $k \in \{\mathbf{b}, \mathbf{g}, \mathbf{r}, \mathbf{a}\}$ del píxel en la fila x y la columna y de la imagen.

Llamaremos O a la imagen de salida generada por cada filtro. Por ejemplo, el filtro identidad estaría caracterizado por la fórmula

$$\forall k \in \{\mathbf{r}, \mathbf{g}, \mathbf{b}, \mathbf{a}\} \quad O[i, j, k] = I[i, j, k].$$

2. Filtros

2.1. Diferencia de imágenes

El siguiente filtro que programaremos generará una imagen resaltando dónde difieren dos imágenes. Cada píxel de la imagen resultante indicará, con un color en escala de grises, cuán

distintos son los píxeles correspondientes de las imágenes de entrada. Para cuantificar esto último tomamos la norma infinito de la resta vectorial entre los píxeles, ignorando la componente a .¹

$$\begin{aligned}\forall k \in \{r, g, b\} \quad O[i, j, k] &= \|I_1[i, j] - I_2[i, j]\|_\infty \\ O[i, j, a] &= 255\end{aligned}$$

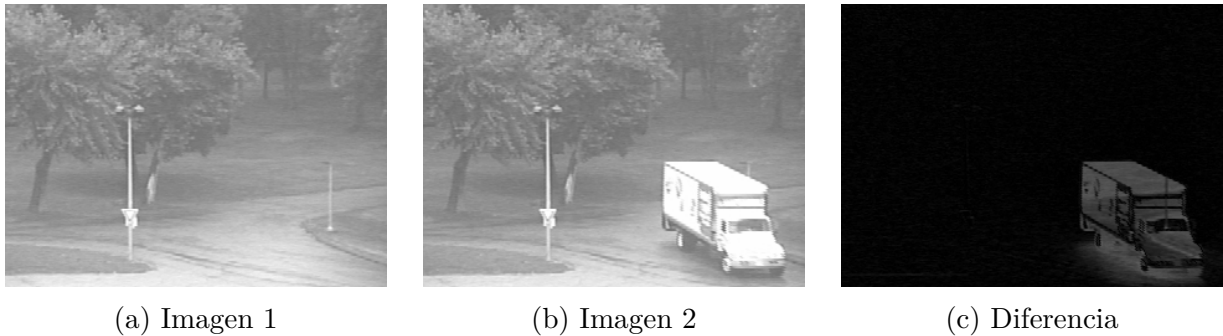


Figura 1: Ejemplo de uso del filtro para detección de primer plano.

2.2. Blur Gaussiano

Se quiere “suavizar” una imagen, es decir darle un aspecto desenfocado. Una manera de lograr este efecto es calculando cada componente de la imagen de salida como un simple promedio de una vecindad del mismo en la imagen original; esto es conocido como *box blur*, y produce un desenfocado un poco “cuadrado”. Tomando un promedio ponderado, en cambio, el blur Gaussiano logra un desenfocado más natural.



Figura 2: Filtro Blur Gaussiano sobre la imagen Lena. ($\sigma = 5$, $r = 15$)

Este promedio ponderado es, de manera más general, una *convolución* aplicada a la matriz imagen. Esta es una herramienta muy usada en procesamiento de imágenes y señales, pues muchos filtros se expresan como convoluciones.

¹ $\|(r, g, b, a)\|_\infty := \max\{|r|, |g|, |b|\}$

Una convolución es una operación matemática que transforma dos funciones f y g en una tercera función, y se denota con el símbolo $*$.

$$(f * g)(t) := \int_{-\infty}^{+\infty} f(\tau) g(t - \tau) d\tau.$$

En el procesamiento digital de señales las funciones involucradas en una convolución no son continuas sino discretas: sólo se tienen muestreos p y q de las funciones en un dominio discreto. En este caso la convolución se puede expresar como una sumatoria.

$$(p * q)[n] = \sum_{k=-\infty}^{\infty} p[k] q[n - k] \quad (n, k \in \mathbb{Z})$$

Aquí p y q pueden ser, por ejemplo, muestreos de señales de audio en función del tiempo. Como en este trabajo nos compete el procesamiento de imágenes, trabajaremos con convoluciones discretas en dos dimensiones, donde una de las funciones involucradas representará a la imagen y la otra caracterizará al filtro que se está aplicando. Además la sumatoria será sobre un dominio acotado porque para el cálculo de cada píxel de salida nos limitaremos a tomar sólo una cierta vecindad (de radio r) de los píxeles de entrada. Realizamos la convolución por separado para cada componente k .

$$O[i, j, k] = (I * K)[i, j, k] = \sum_{x=-r}^r \sum_{y=-r}^r I[i + x, j + y, k] K[r - x, r - y] \quad (1)$$

La matriz K recibe el nombre de matriz de convolución o *kernel*. Según esta fórmula, el píxel $[i, j]$ en la imagen de salida O se calculará como la suma de los píxeles en la vecindad de $[i, j]$ de la imagen de entrada I , ponderada según los pesos en la matriz de convolución. La figura 3 ilustra el cálculo de un píxel.

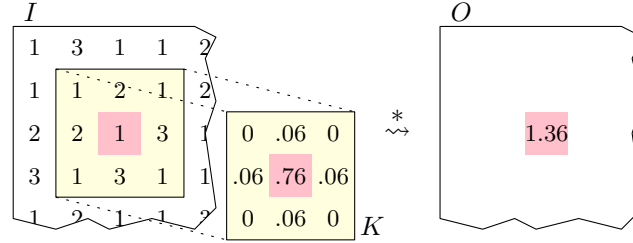


Figura 3: Ejemplo de cálculo de un píxel en una convolución.

En el blur Gaussiano, la matriz de convolución se calcula usando una función Gaussiana parametrizada por σ , la desviación estándar deseada, que determina cuánto se suaviza la imagen. Dado que dicha función integra a 1, decimos que esta convolución es un *promedio* ponderado; esto garantiza que el brillo de la imagen no se altera.

$$K_{\sigma,r}[i, j] = G_{\sigma}(r - i, r - j) \quad G_{\sigma}(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Notemos que el valor en los píxeles del borde de la imagen no estará definido (porque no hay suficientes vecinos), así que los dejaremos intactos en la imagen de salida. Con excepción

de estos, el resto de los píxeles deberán calcularse como la convolución de la imagen con la matriz Gaussiana:

$$\forall k \in \{r, g, b\} \quad O[i, j, k] = \sum_{x=-r}^r \sum_{y=-r}^r I[i+x, j+y, k] K_{\sigma, r}[r-x, r-y]$$

$$O[i, j, a] = 255$$

El filtro está parametrizado por σ y por r . Con lo cual deberán implementar por un lado el cálculo de la matriz de convolución, y por otro lado la convolución propiamente dicha. Lo primero puede estar escrito en C o en assembler según les parezca conveniente.

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 tres tipos de formatos, tanto con o sin transparencia. Ambos formatos corresponden a los tipos de encabezado: BITMAPINFOHEADER (40 bytes), BITMAPV3INFOHEADER (56 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 líneas 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. Dentro de cada línea los pixeles se almacenan de izquierda a derecha, y cada pixel **en memoria se guarda en el siguiente orden: B, G, R, A**.

4. Ejercicios

Los ejercicios están divididos en dos partes, por un lado la implementación de los filtros y por otro el análisis de los mismos.

Implementación

Deberán implementar (al menos) una versión de cada filtro en C y otra en lenguaje ensamblador, utilizando instrucciones SSE.

Análisis

Consiste en estudiar los dos filtros propuestos con sus distintas implementaciones.

1. Comparación entre ASM_blur y C_blur
2. Comparación entre ASM_diff y C_diff

Las siguientes preguntas deben ser usadas como guía. La evaluación del trabajo práctico no sólo consiste en responder las preguntas, sino en desarrollar y responder nuevas preguntas sugeridas por ustedes mismos buscando entender y razonar sobre el modelo de programación SIMD y la microarquitectura del procesador.

- ¿Qué métricas se pueden utilizar para calificar las implementaciones y cuantificarlas?
- ¿Cuál implementación es mejor?
- ¿En qué casos? ¿De qué depende? ¿Depende del tamaño de la imagen? ¿Depende de la imagen en sí?
- ¿Cómo se podrían mejorar las métricas de las implementaciones propuestas? ¿Cuáles no se pueden mejorar?
- ¿Es una comparación justa? ¿La cantidad de operaciones en cada implementación es la misma? ¿Y los accesos a memoria?
- ¿Las limitaciones en las métricas son por los accesos a memoria?, ¿o a la memoria cache?, ¿esta se podría acceder mejor?
- ¿Y los saltos condicionales? ¿Es posible evitarlos?
- ¿El patrón de acceso a la memoria es desalineado? ¿Hay forma de mejorarlo? ¿Es posible medir cuánto se pierde?
- ¿Hay diferencias en operar con enteros o punto flotante? ¿La imagen final tiene diferencias significativas?
- ¿El overhead de llamados a funciones es significativo? ¿Se puede medir?

4.1. 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 ni tener overflow 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.

4.2. 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:

- **codigo**: Contiene los fuentes del programa principal **tp2**. Dentro de la misma encontramos:
 - **helper**: Contiene los fuentes de la biblioteca de BMP y los fuentes del programa **diff**
 - **filtros**: Contiene las implementación de todos los filtros
 - **build**: Contiene los archivos **.o**, los ejecutables (**tp2** y **diff**)
 - **img**: Contiene imágenes de prueba

4.2.1. Compilar

Ejecutar **make** desde la carpeta **src**.

4.2.2. Uso

El siguiente es el uso básico del ejecutable **tp2**, pero los invitamos a ejecutar **tp2 -h** para ver la totalidad de las opciones disponibles. En general lo deberán ejecutar como **./build/tp2**.

```
tp2 -i <c/asm> <filtro> <parametros...>
```

La opción **-i** define la implementación a utilizar, puede ser **c** o **asm**. El primer parámetro es el nombre del filtro, puede ser **blur** o **diff**. Por último los parámetros adicionales dependen del filtro.

```
blur <src> <sigma> <radio>
diff <src1> <src2>
```

- **src**: Ruta del archivo de entrada
- **sigma**: valor entre 0 y 1 que indica el parámetro σ para el blur Gaussiano.
- **radio**: entero que indica la distancia máxima al centro de la matriz de convolución.

4.2.3. Ejemplo de uso

- **./build/tp2 -i asm blur img/lena.bmp 0.78 2**

Aplica el filtro **blur Gaussiano** a la imagen **../img/lena.bmp** con $\sigma = 0,78$ en una matriz de 5x5 ($r = 2$ pixeles de distancia al centro de la matriz) utilizando la implementación en **asm** del filtro y almacena el resultado en el directorio actual.

4.2.4. 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 después 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 varían 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. En el archivo `rdtsc.h` encontrarán las funciones necesarias para implementarla.

4.2.5. Herramientas

En el código provisto, podrán encontrar una herramienta que les permitirá comparar dos imágenes. El código de la misma se encuentra en `helper`, y se compila junto con el resto del trabajo práctico. El ejecutable, una vez compilado, se almacenará en `build/bmpdiff`. La aplicación se utiliza desde línea de comandos de la forma:

```
./build/bmpdiff <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 imágenes blanco y negro por cada componente, donde blanco es marca que hay diferencia y negro que no.

Las opciones soportadas por el programa son:

| | |
|----------------------------|--|
| <code>-i, --image</code> | Genera imágenes de diferencias por cada componente. |
| <code>-v, --verbose</code> | Lista las diferencias de cada componente y su posición en la imagen. |
| <code>-a, --value</code> | Genera las imágenes mostrando el valor de la diferencia. |
| <code>-s, --summary</code> | Muestra un resumen de diferencias. |

4.3. 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 descripción 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 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 una o varias hipótesis y experimentos que les permitan comprobarlas o refutarlas. Estas hipótesis y los experimentos deberán ser aprobadas por sus correctores. 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.

5. Entrega

Se deberá entregar un archivo comprimido que contendrá la carpeta raíz junto con el informe.

La fecha límite para la presentación de hipótesis y experimentos es el **17/9**. Deberá ser enviada por correo electrónico en texto plano de no mas de 400 palabras a **orga2-doc@dc.uba.ar** con el asunto **[TP2-Experimentos]**. No olviden incluir el nombre y el número de libreta de cada integrante del grupo. No serán corregidos trabajos que hayan sido entregados sin que los experimentos hayan sido aprobados previa a la entrega.

La fecha de entrega de este trabajo es **29/9** y deberá ser entregado a través de la página web. El sistema sólo aceptará entregas de trabajos hasta las **17:00hs** del día de entrega.

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