Trabajo Práctico 2

Organización del Computador II

Segundo Cuatrimestre de 2016

1. Introducción

En este trabajo práctico buscamos una primera aproximación al modelo de procesamiento SIMD. 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.

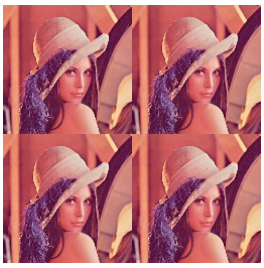
Como campo de aplicación tomamos el procesamiento de imágenes. Deberán implementar varios filtros, cada uno de ellos en C y en lenguaje ensamblador, para luego plantear hipótesis, experimentar y sacar conclusiones respecto de cada implementación.

Esto último debe llevarse a cabo con un carácter científico y con las metodologías correspondientes, tomando como factor de mayor importancia la rigurosidad y exhaustividad del análisis que realicen. Además dedicaremos una clase práctica a estos temas.

2. Filtros

Los filtros a implementar se describen a continuación. Aquí una imagen de cada uno a modo de ejemplo.

Imagen original



Smalltiles

Rotar

Pixelar

Combinar

Colorizar

2.1. 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 cada una de estas componentes tendrá 8 bits (1 byte) de profundidad, es decir que estarán representadas por números enteros en el rango $[0, 256)$.

Dada una imagen I, notaremos $\mathsf{I}^k_{i,j}$ al valor de la componente $k \in \{\mathsf{r}, \mathsf{g}, \mathsf{b}, \mathsf{a}\}$ del píxel en la fila i y la columna j de la imagen. La fila 0 corresponde a la fila de más abajo de la imagen. La columna 0 a la de más a la izquierda.

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

$$\forall k \in \{\mathsf{r}, \mathsf{g}, \mathsf{b}, \mathsf{a}\} \quad \mathsf{O}^k_{i,j} = \mathsf{I}^k_{i,j}.$$

2.2. Smalltiles

Esta operación consiste en repetir la imagen original 4 veces, de forma más chica en la imagen destino. Es decir, que si originalmente se tiene 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.

Implementación y uso

Funciones: smalltiles_c, smalltiles_asm
Parámetros: ninguno

2.3. Rotar canales

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

$$\mathbf{R} \longrightarrow \mathbf{G}$$
$$\mathbf{G} \longrightarrow \mathbf{B}$$
$$\mathbf{B} \longrightarrow \mathbf{R}$$

Funciones: rotar_c, rotar_asm
Parámetros: ninguno

2.4. Pixelar

El proceso de pixelar la imagen consiste en partir la imagen original en bloques de 2×2 píxeles. Por cada componente por separado, y para cada bloque de la imagen original, se genera un bloque del mismo tamaño en la imagen destino y a cada uno de sus píxeles se le asigna el promedio de los valores de los píxeles del bloque de la imagen original.

Nota: El tamaño (tanto alto como ancho) de la imagen destino es múltiplo de 4.

Implementación y uso

Funciones: pixelar_c, pixelar_asm
Parámetros: ninguno

2.5. Combinar

Dadas 2 imágenes de igual tamaño, este procedimiento genera una tercera formada a partir de estas 2. Cada píxel de la imagen resultante se forma de la siguiente manera:

$$I_{dst}(i,j) = \frac{alpha \cdot (I_{src_a}(i,j) - I_{src_b}(i,j))}{255{,}0} + I_{src_b}(i,j)$$

donde $alpha \in [0{,}0; 255{,}0]$.

Nota: Por simplicidad, este proceso se realiza con la imagen original y su reflejo vertical.

Implementación y uso

Funciones: combinar_c, combinar_asm,
Parámetros:

- alpha: Número en punto flotante entre 0.0 y 255.0

Ejemplo de uso: combinar -i c lena.bmp

2.6. Colorizar

Dada una imagen de entrada, la imagen resultado se forma en base a las siguientes definiciones:

$$\begin{aligned} max_*(i,j) = max(\quad & I_{src\_*}(i-1,j-1) \quad , \quad I_{src\_*}(i-1,j) \quad , \quad I_{src\_*}(i-1,j+1), \\ & I_{src\_*}(i,j-1) \quad , \quad I_{src\_*}(i,j) \quad , \quad I_{src\_*}(i,j+1), \\ & I_{src\_*}(i+1,j-1) \quad , \quad I_{src\_*}(i+1,j) \quad , \quad I_{src\_*}(i+1,j+1)) \end{aligned}$$

donde $* \in \{R, G, B\}$. Luego

$$\phi_R(i,j) = \begin{cases} (1+\alpha) & \text{si } max_R(i,j) \geq max_G(i,j) \text{ y } max_R(i,j) \geq max_B(i,j) \\ (1-\alpha) & \text{si no} \end{cases}$$

$$\phi_G(i,j) = \begin{cases} (1+\alpha) & \text{si } max_R(i,j) < max_G(i,j) \text{ y } max_G(i,j) \geq max_B(i,j) \\ (1-\alpha) & \text{si no} \end{cases}$$

$$\phi_B(i,j) = \begin{cases} (1+\alpha) & \text{si } max_R(i,j) < max_B(i,j) \text{ y } max_G(i,j) < max_B(i,j) \\ (1-\alpha) & \text{si no} \end{cases}$$

donde $0 \leq \alpha \leq 1$.

$$
\begin{aligned}
I_{dst_R}(i,j) &= min(255, \phi_R * I_{src_R}(i,j)) \\
I_{dst_G}(i,j) &= min(255, \phi_G * I_{src_G}(i,j)) \\
I_{dst_B}(i,j) &= min(255, \phi_B * I_{src_B}(i,j))
\end{aligned}
$$

Nota: La primera y última fila de la imagen original no debe ser procesada. Lo mismo sucede para la primera y última columna.

Implementación y uso

Funciones: colorizar_c, colorizar_asm
Parámetros:

- float alpha: número entre 0 y 1 que define intensidad de la colorización

3. Implementación

Para facilitar el desarrollo del trabajo práctico se cuenta con un *framework* que provee todo lo necesario para poder leer y escribir imágenes, así como también compilar y probar las funciones que vayan a implementar.

3.1. Archivos y uso

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.

Los archivos entregados están organizados en las siguientes carpetas:

- documentos: Contiene este enunciado y un *template* de informe en LaTeX.
- codigo: Contiene el código fuente, junto con el framework de ejecución y testeo. Contiene los fuentes del programa principal, junto con el **Makefile** que permite compilarlo. Además contiene los siguientes subdirectorios:
  - build: Contiene los archivos objeto y ejecutables del TP.
  - filtros: Contiene las implementaciones de los filtros
  - helper: Contiene los fuentes de la biblioteca BMP y de la herramienta de comparación de imágenes.
  - img: Algunas imágenes de prueba.
  - test: Contiene scripts para realizar tests sobre los filtros y uso de la memoria.

Compilación

Ejecutar `make` desde la carpeta `codigo`. Recordar que cada directorio tiene su propio Makefile, por lo que si se desea cambiar las opciones de compilación debe buscarse el Makefile correspondiente.

Uso

El uso del programa principal es el siguiente:

```
$ ./tp2 <opciones> <nombre_filtro> <nombre_archivo_entrada> [parámetros...]
```

Los filtros que se pueden aplicar y sus parámetros son los especificados en la sección filtros, apartado "Implementación y uso"

Las opciones que acepta el programa son las siguientes:

- -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
- -o, –output DIRECTORIO
  Genera el resultado en DIRECTORIO. De no incluirse, el resultado se guarda en el mismo directorio que el archivo fuente
- -v, –verbose
  Imprime información adicional

Por ejemplo:

```
$ ./tp2 -v colorizar -i asm lena.bmp 0.4
```

Aplica el filtro de **colorizar** al archivo lena.bmp utilizando la implementación en lenguaje asm del filtro, pasándole como parámetro 0.4 como valor de alpha.

3.2. Código de los filtros

Para implementar los filtros descriptos anteriormente, tanto en C como en ASM se deberán implementar las funciones especificadas en la sección 2. Las imágenes se almacenan en memoria en color, en el orden B (blue), G (green), R (red), A (alpha).

Los parámetros genéricos de las funciones son:

- src : Es el puntero al inicio de la matriz de elementos de 32 bits sin signo (el primer byte corresponde al canal azul de la imagen (B), el segundo el verde (G), el tercero el rojo (R)), y el cuarto el alpha (A) que representa a la imagen de entrada. Es decir, como la imagen está en color, cada píxel está compuesto por 4 bytes.
- dst : Es el puntero al inicio de la matriz de elementos de 32 bits sin signo que representa a la imagen de salida.
- filas : Representa el alto en píxeles de la imagen, es decir, la cantidad de filas de las matrices de entrada y salida.
- cols : 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 el ancho en bytes de cada fila de la imagen incluyendo el padding en caso de que hubiere. Es decir, la cantidad de bytes que hay que avanzar para moverse a la misma columna de fila siguiente/anterior.

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, a menos que se indique lo contrario.
- 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 2 píxeles simultámeamente**.

  De no ser posible esto para algún filtro, deberá justificarse debidamente en el informe.
- El procesamiento de los píxeles se deberá hacer **exclusivamente** con instrucciones **SSE**. No está permitido procesarlos con registros de propósito general, salvo para tratatimiento de casos borde. En tal caso se deberá justificarse debidamente y hacer un análisis del costo computacional.
- El TP se tiene que poder ejecutar en las máquinas del laboratorio.

3.3. 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**.

3.4. Herramientas y tests

En el código provisto, podrán encontrar varias herramientas que permiten verificar si los filtros funcionan correctamente.

Diff

La herramienta diff permite 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 linea 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:

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

Tests

Para verificar el correcto funcionamiento de los filtros, además del comparador de imágenes, se provee un binario con la solución de la cátedra y varios scripts de test. El binario de la cátedra se encuentra en la carpeta codigo/build. El comparador de imágenes se ubica en la carpeta codigo/helper, y debe compilarse antes de correr los scripts (correr make en la carpeta codigo/helper).

Los scripts de test toman como entrada las corridas especificadas en corridas.txt. Para cada imagen de test, se ejecutan todas las corridas ahí indicadas. Para verificar que la implementación funciona correctamente con imágenes de distinto tamaño, generar_imagenes.sh genera variaciones de las imágenes fuente (que se encuentran en codigo/tests/data), y las deposita en imagenes_a_testear. Para que este script funcione correctamente se requiere la utilidad convert que se encuentra en la biblioteca imagemagick.[1].

El archivo test_dif_cat.sh verifica que los resultados de la catedra den igual que la implementación de C. test_dif_c_asm.sh verifica que los resultados de las versiones de C y Assembler sean iguales. test_mem.sh chequea que no haya problemas en el uso de la memoria. Finalmente, test_all.sh corre todos los checks anteriores uno después del otro.

3.5. 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 distintas problemáticas a solucionar:

a) 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.

[1] Para instalar sudo apt-get install imagemagick

b) Los procesadores modernos varían su frecuencia de reloj, por lo que la forma de medir ciclos cambiará dependiendo del estado del procesador.

c) El comienzo y fin de la medición deben realizarse con la suficiente exactitud como para que se mida solamente la ejecución de los filtros, sin ser afectada por ruidos como la carga o el guardado de las imágenes.

Para medir tiempos deberán idear e implementar una metodología que les permita evitar estos tres problemas. En el archivo `tp2.c` se provee código para realizar una medición de tiempo básica. El mismo podrá ser modificado para mejorar y automatizar las mediciones. Se recomienda utilizar un framework de medición automatizado como metrika[2] para evitar errores de medición sistemáticos, es decir, aquellos causados por una incorrecta ejecucción de las mediciones.

4. Ejercicios

Se deberá implementar el código de los filtros, realizar un análisis de su performance y presentar un informe de los resultados.

4.1. Implementación

Deberán implementar (al menos) una versión de cada filtro en C y otra en lenguaje ensamblador, utilizando instrucciones SSE. La implementación inicial de los filtros que sólo realicen cálculos con números enteros no deberá perder precisión. Es posible que se desee realizar optimizaciones al costo de perder algo de precisión. Esto será aceptable siempre y cuando se analice también la calidad de la imagen resultante en la versión optimizada contra la no optimizada.

4.2. Análisis

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.

- ¿Cuál implementación es "mejor"?
- ¿Qué métricas se pueden utilizar para calificar las implementaciones y cuantificarlas?
- ¿En qué casos? ¿De qué depende? ¿Depende del tamaño de la imagen? ¿Depende de la imagen en sí? ¿De los parámetros?
- ¿Cómo se podrían mejorar las métricas de las implementaciones propuestas? ¿Cuáles no se pueden mejorar?
- ¿Es una comparación justa? ¿De qué depende la velocidad del código C? ¿Cómo puede optimizarse?

[2] pip3 install –user metrika — `https://github.com/dc-uba/metrika`

- ¿Cuál es la cantidad de instrucciones ejecutadas por pixel en cada implementación? ¿Y de accesos a memoria? ¿Se condice empíricamente esta diferencia en la performance de los filtros?
- ¿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?
- ¿Las limitaciones de performance son causadas por los accesos a memoria?, ¿o a la memoria cache?, ¿esta se podría acceder mejor?
- ¿Y los saltos condicionales? ¿Afectan la performance? ¿Es posible evitarlos total o parcialmente?
- ¿El patrón de acceso a la memoria es desalineado? ¿Hay forma de mejorarlo? ¿Es posible medir cuánto se pierde?

4.3. Informe

El informe debe incluir las siguientes secciones:

a) **Carátula** Contiene

- número / nombre del grupo
- nombre y apellido de cada integrante
- número de libreta y mail de cada integrante

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

c) **Desarrollo**
Describe 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 a los registros, cómo los reordenan para procesarlos, las operaciones que se aplican a los datos, etc. Además se agregará un detalle más profundo de las secciones de código que consideren más importantes. 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 al procesamiento simultáneo SIMD. 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 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 comprobar las diferencias de performance e hipotetizar sobre sus causas.

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.

Utilizar como guía para la realización de experimentos las preguntas de la sección anterior. Al responder estas preguntas (y otras que vayan surgiendo), se deberán analizar y comparar las implementaciones de cada funcion en su versión `C` y `ASM`, mostrando los resultados obtenidos a través de tablas y gráficos. También se deberá *comentar* los resultados obtenidos.

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

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 y condiciones de aprobación

El presente trabajo es de carácter **grupal**, siendo los grupos de **3 personas**, pudiendo ser de 2 personas en casos excepcionales previa consulta y confirmación del cuerpo docente. 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. **No** debe incluírse ningún tipo de archivo binario extra como código objeto o ejecutables.

La fecha de entrega última de este trabajo es **Martes 27 de Septiembre** 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. La fecha límite para la **reentrega** es el día **Martes 1 de Noviembre**

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