Taller 4

Organización del Computador II

Primer Cuatrimestre de 2023

1. Introducción

Este taller consiste en resolver ejercicios e implementar filtros gráficos utilizando el modelo de procesamiento SIMD. El mismo consta de dos partes. En primer lugar, resolverán dos ejercicios para empezar a familiarizarse con las instrucciones del modelo SIMD. En segundo lugar, aplicaremos lo estudiado en clase y lo practicado previamente programando de manera vectorizada un conjunto de filtros para imágenes.

2. Ejercicios

2.1. Ejercicio 1

Deberán implementar la función invertirBytes, la cual es parte de un sistema de detección y corrección de errores. El prototipo de la función es:

```
void invertirBytes_asm(uint8_t* p, uint8_t n, uint8_t m)
```

La función a implementar será llamada por el detector de errores cuando este detecte que los valores de dos bytes dentro de una posición de memoria de 128 bits están invertidos. La función recibirá como parámetros un puntero (p) que apunta a la posición de memoria donde se almacenan 128 bits y dos enteros de 8 bits (n y m) indicando cuales son los bytes que deben ser invertidos de lugar en la posición de memoria apuntada por p. Notar que los valores de n y m irán de 0 a 15 y que n puede ser mayor, menor o igual que m.
Para ejemplificar, si n = 2 y m = 11 la memoria apuntada por p antes de ejecutar la función debería verse como:

0	1	n	3	4	5	6	7	8	9	10	m	12	13	14	15

Y luego de ejecutar la función debería quedar:

0	1	m	3	4	5	6	7	8	9	10	n	12	13	14	15

Es decir, se intercambian los bytes n y m. **La función debe implementarse en asm utilizando instrucciones del modelo SIMD.**

La siguiente información aplica para este checkpoint y el siguiente

Compilación y Testeo

El archivo `main.c` es para que ustedes realicen pruebas básicas de sus funciones. Siéntanse a gusto de manejarlo como crean conveniente. Para compilar el código y poder correr las pruebas cortas implementadas en main deberá ejecutar `make main` y luego `./runMain.sh`.
En cambio, para compilar el código y correr las pruebas intensivas deberá ejecutar `./runTester.sh`.
El programa puede correrse con `./runMain.sh` para verificar que no se pierde memoria ni se realizan accesos incorrectos a la misma.

Pruebas intensivas (Testing)

Entregamos también una serie de *tests* o pruebas intensivas para que pueda verificarse el buen funcionamiento del código de manera automática. Para correr el testing se debe ejecutar `./runTester.sh`, que compilará el *tester* y correrá todos los tests de la cátedra. Luego de cada test, el *script* comparará los archivos generados por su TP con la solución en C provista por la cátedra. También será probada la correcta administración de la memoria dinámica.

Checkpoint 1

2.2. Ejercicio 2

En este ejercicio deben implementar una función para detectar errores. Esta función recibe un puntero a un arreglo (notar que no es un arreglo tradicional de C) y la cantidad de ternas que recibe. El prototipo de la función es:

```
uint8_t checksum_asm(void* array, uint32_t n)
```

El arreglo estará formado de la siguiente manera:

Donde A y B son enteros sin signo de 16 bits y C son enteros sin signo de 32 bits. Además, como podrán notar A,B y C están alternados, luego de 8 valores de A, siguen 8 valores de B y a continuación 8 valores de C y esto se repite hasta llegar a n ternas ABC. Se garantiza que el arreglo en memoria contenerá cómo mínimo 1 terna.

Esta función deberá chequear que se cumpla lo siguiente:

$$C_{ji} = (A_{ji} + B_{ji}) * 8 \tag{1}$$

La función devolverá 1 si se cumple la condición para todo el arreglo y 0 en caso contrario.

Tener en cuenta que deben hacer uso del modelo SIMD para implementar esta función.

Checkpoint 2

3. Filtros

Deberán implementar 2 filtros diferentes. La implementación de los filtros que sólo realicen cálculos con números enteros no podrá perder precisión. Las explicación detallada del taller se encuentra después de los checkpoints.

- Deberán implementar el filtro Offset en lenguaje ensamblador, utilizando instrucciones SIMD y procesando como mínimo 4 píxeles en simultáneo. Comparar y discutir la cantidad de ciclos que le tomo a las implementaciones en c y asm para ejecutar el filtro.

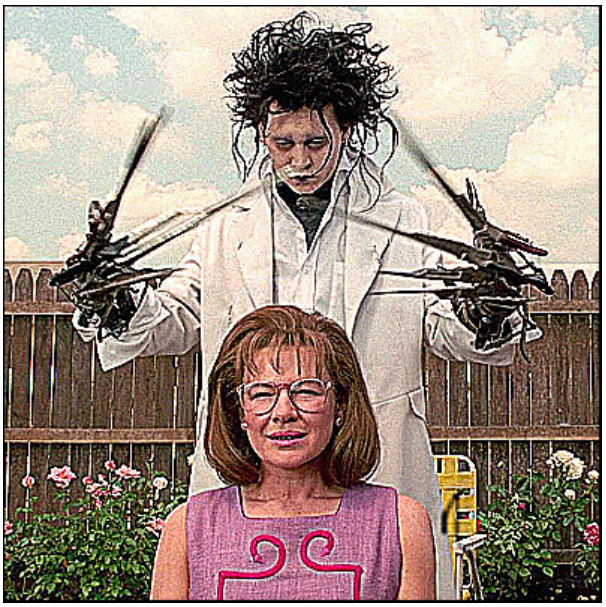
Checkpoint 3

- Deberán implementar el filtro Sharpen en lenguaje ensamblador, utilizando instrucciones SIMD y procesando como mínimo 2 píxeles en simultáneo. Comparar y discutir la cantidad de ciclos que le tomo a las implementaciones en c y asm para ejecutar el filtro.

Checkpoint 4

Original

Offset



Sharpen

3.1. 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, estarán representadas por números enteros en el rango $[0, 255]$.

Nombraremos a las matrices de píxeles `src` como la imagen de entrada y como `dst` a la imagen destino. Ambas serán almacenadas por filas, tomando el píxel *[0,0]* de la matriz como el píxel arriba a la izquierda de la imagen. Los filtros reciben un puntero a la imagen que apunta al pixel de arriba a la izquierda y se la recorre izquierda a derecha, arriba a abajo, avanzando la dirección de memoria del puntero.[1]

En todos los filtros, el valor de la componente de transparencia debe ser 255.

3.2. Offset

El color de cada píxel está dado por sus tres componentes: rojo, verde y azul. En las impresiones por offsets, los colores se forman aplicando una a una cada componente sobre el papel. Cuando la aplicación de los colores se desfasa, se produce un efecto de estela de color alrededor de cada figura. El objetivo de este filtro es generar este efecto para un desfasaje fijo de las componentes de color de las imágenes, respetando el siguiente fragmento de código en C:

```
for (int i = 8; i < height-8; i++) {
   for (int j = 8; j < width-8; j++) {

        dst[i][j].b = src[i+8][j].b;
        dst[i][j].g = src[i][j+8].g;
        dst[i][j].r = src[i+8][j+8].r;
    }
}
```

[1]Notar que el formato BMP utilizado para almacenar las imágenes, invierte el orden de las filas en el archivo almacenado.

Las operaciones se realizan dejando un marco de 8 píxeles alrededor de toda la imagen. Estos píxeles deben ser completados con el color negro (0,0,0) sobre la imagen destino.

3.3. Sharpen

Existe un conjunto de efectos sobre imágenes que se realizan aplicando operaciones matriciales sobre los píxeles. Estas matrices se denominan operadores o *kernels*. Para este filtro se pide implementar el operador *Sharpen* o "realzado" respetando el siguiente pseudocódigo:

```
                        | -1, -1, -1 |
float sharpen[3][3] = | -1,  9, -1 |
                        | -1, -1, -1 |

Para i de 0 a height - 3:
    Para j de 0 a width - 3:

        totalB = 0, totalG = 0, totalR = 0

        Para ii de 0 a 2:
            Para jj de 0 a 2:
                totalB = totalB + sharpen[ii][jj] * src[i+ii][j+jj].b
                totalG = totalG + sharpen[ii][jj] * src[i+ii][j+jj].g
                totalR = totalR + sharpen[ii][jj] * src[i+ii][j+jj].r

        dst[i+1][j+1].b = SAT(totalB);
        dst[i+1][j+1].g = SAT(totalG);
        dst[i+1][j+1].r = SAT(totalR);
```

Las operaciones se realizan dejando un marco de 1 píxel alrededor de toda la imagen. Estos píxeles deben ser completados con el color negro (0,0,0) sobre la imagen destino.

4. Implementación

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

4.1. Archivos y uso

Dentro de los archivos presentados deben completar el código de las funciones pedidas. Puntualmente encontrarán el programa principal, denominado **simd**, 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:

- doc: Contiene este enunciado
- src: Contiene el código fuente, junto con el framework de ejecución y testeo. También tiene los fuentes del programa principal, junto con el **Makefile** que permite compilarlo. Además posee los siguientes subdirectorios:
    - build: Los archivos objeto y ejecutables del taller.

- filters: Las implementaciones de los filtros
- helper: Los fuentes de la biblioteca BMP y de la herramienta de comparación de imágenes.
- img: Algunas imágenes de prueba.
- test: Scripts para realizar tests sobre los filtros y uso de la memoria.

Compilación

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

Uso

El uso del programa principal es el siguiente:

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

Los filtros que se pueden aplicar y sus parámetros son los especificados en la sección **??**.
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. Las 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.

4.2. 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 **??**. 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 hubiese. Es decir, la cantidad de bytes que hay que avanzar para moverse a la misma columna de fila siguiente o anterior.

Consideraciones

Las funciones a implementar en lenguaje ensamblador deben utilizar el set de instrucciones **SIMD**, 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 y múltiplo de 8 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.
- Se debe trabajar con la máxima cantidad de píxeles en simultáneo.
- El procesamiento de los píxeles se deberá hacer **exclusivamente** con instrucciones **SIMD**. No está permitido procesarlos con registros de propósito general, salvo para tratamiento de casos borde. En tal caso se deberá justificar debidamente.

4.3. Formato BMP

Lo que contamos a continuación del formato BMP es sólo informativo. Lo importante es lo que aparece en **negrita**.

El formato BMP es uno de los formatos de imágenes más simples: tiene un encabezado y un mapa de bits que representa la información de los píxeles.

En este taller 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 provisto por la cátedra. 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 píxeles se almacenan de izquierda a derecha, y cada píxel **en memoria se guarda en el siguiente orden: B, G, R, A**.

Tengan en cuenta que las imágenes que se pasan a los filtros que tienen que implementar, se pasan tomando como primera línea la que se encuentra arriba en la imagen.

4.4. Herramientas y tests

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

bmpdiff

La herramienta bmpdiff permite comparar dos imágenes. El código de la misma se encuentra en `helper`, y se compila junto con el resto del taller. El ejecutable, una vez compilado, se almacenará en `build/bmpdiff`.

La aplicación se utiliza desde la línea de comandos de la forma:

```
$ ./build/bmpdiff [opciones] <archivo_1> <archivo_2> <epsilon>
```

Compara los dos archivos según las componentes de cada píxel, siendo `epsilon` la diferencia máxima permitida entre píxeles correspondientes a las dos imágenes. Tiene dos opciones: listar las diferencias o generar imágenes blanco y negro por cada componente, donde blanco 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 funcionamiento correcto de los filtros, además del comparador de imágenes, se provee de un binario con la solución de la cátedra y varios scripts de test. El binario de la cátedra se encuentra en la carpeta test/.

El comparador de imágenes se ubica en la carpeta src/helper, y debe compilarse antes de correr los scripts.

Para ejecutar los script, en primer lugar, se deben generar las imagenes de prueba. Esto se realiza ejecutando el script `1_generar_imagenes.py`. Para que este script funcione correctamente se requiere la utilidad convert que se encuentra en la biblioteca imagemagick.[2].

Luego, el script `2_test_diff_cat_asm.py`, chequea diferencias entre las imagenes generadas por el binario de la cátedra y su binario. Este script prueba solamente por diferencias el resultado final.

Por último, el script `3_correr_test_mem.sh`, chequea el correcto uso de la memoria. Al demorar mucho tiempo, este test se ejecuta solamente para los casos más chicos.

[2]Para instalar sudo apt-get install imagemagick