Trabajo Práctico 2

Organización del Computador II

Primer Cuatrimestre 2014

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 y video.

En este trabajo práctico se implementarán filtros para estas aplicaciones, utilizando lenguaje ensamblador (ASM) e instrucciones **SSE**, y se analizará la performance del procesador al hacer uso de **SIMD** para estas aplicaciones, comparándolas con sus implementaciones respectivas en lenguaje **C**.

Para cada filtro se entrega una descripción matemática exacta. Notar, para las optimizaciones, que no es necesario seguir el procedimiento al pie de la letra. Lo importante es que el resultado final sea el mismo que el dado por la descripción matemática.

2. Filtros

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

Imagen original

Tiles

Popart

Temperatura

LDR

2.1. Filtro de tiles

El filtro de tiles se aplica pixel a pixel en una imágen en color. La función recibe cuatro argumentos que representan un rectángulo dentro de la imagen fuente: tamx, tamy, offsetx, offsety.

Usando esto, el filtro de color escribe repetidamente ese rectángulo en la imagen destino. Su descripción matemática está dada por la siguiente función:

i y j representan fila y columna respectivamente.

$$\mathsf{dst}_{(i,j)} = \mathsf{src}_{[(i \ mod \ \mathsf{tamy})+\mathsf{offsety}][(j \ mod \ \mathsf{tamx})+\mathsf{offsetx}]}$$

2.2. Popart

El filtro popart toma una imagen fuente y genera un efecto que simula el estilo *popart*[1]. El filtro toma los tres componentes del cada pixel, y los suma entre si.

$$\mathsf{suma}_{(i,j)} = (\mathsf{src}.r_{(i,j)} + \mathsf{src}.g_{(i,j)} + \mathsf{src}.b_{(i,j)})$$

El resultado se divide en 5 bandas, cada una de las cuales se mapea a un color diferente.

$$\mathsf{dst}_{(i,j)} < r, g, b >= \begin{cases} < 0, 0, 255 > & \text{si suma}(i,j) < 153 \\ < 127, 0, 127 > & \text{si } 153 \leq \text{suma} < 306 \\ < 255, 0, 255 > & \text{si } 306 \leq \text{suma} < 459 \\ < 255, 0, 0 > & \text{si } 459 \leq \text{suma} < 612 \\ < 255, 255, 0 > & \text{si no} \end{cases}$$

2.3. Temperatura

El filtro temperatura toma una imagen fuente y genera un efecto que simula un mapa de calor. El filtro toma los tres componentes del cada pixel, los suma y divide por 3, y califica a eso como la temperatura t.

$$\mathsf{t}_{(i,j)} = \lfloor(\mathsf{src}.r_{(i,j)} + \mathsf{src}.g_{(i,j)} + \mathsf{src}.b_{(i,j)})/3\rfloor$$

En función de la temperatura, se determina el color en la imagen destino. Para evitar diferencias con los resultados la cátedra, la temperatura debe truncarse y guardarse en una variable de tipo entero.

$$\mathsf{dst}_{(i,j)} < r, g, b >= \begin{cases} < 0, 0, 128 + t \cdot 4 > & \text{si } t < 32 \\ < 0, (t-32) \cdot 4, 255 > & \text{si } 32 \leq t < 96 \\ < (t-96) \cdot 4, 255, 255 - (t-96) \cdot 4 > & \text{si } 96 \leq t < 160 \\ < 255, 255 - (t-160) \cdot 4, 0 > & \text{si } 160 \leq t < 224 \\ < 255 - (t-224) \cdot 4, 0, 0 > & \text{si no} \end{cases}$$

[1] https://www.google.com/images?q=popart

2.4. Low dynamic range

El filtro LDR toma una imagen fuente y aplica un efecto que modifica la imagen según su iluminación. El filtro toma el valor de un pixel y le añade un porcentaje α de el de sus vecinos. De esta manera, dado un porcentaje positivo, los píxeles rodeados por píxeles claros se vuelven aún más claros, mientras que los rodeados por pixeles oscuros se mantienen igual. La intensidad del efecto dependerá del porcentaje sumado.

Para cada componente independiente del pixel (R, G y B) la fórmula matemática será

$$\mathsf{dst}_{(i,j)} = \text{mín}\,\text{máx}\,(\mathsf{src}_{(i,j)} + \mathsf{var}_{(i,j)}, 0, 255)$$

donde

$$\mathsf{var}_{(i,j)} = \mathsf{src}(i,j) \cdot \alpha \frac{\mathsf{sumargb}_{(i,j)}}{\mathsf{max}}$$

$$\mathsf{sumargb}_{(i,j)} = \mathsf{suma}_{\mathsf{r}(i,j)} + \mathsf{suma}_{\mathsf{g}(i,j)} + \mathsf{suma}_{\mathsf{b}(i,j)}$$

$$\mathsf{max} = 5 * 5 * 255 * 3 * 255$$

y finalmente $\mathsf{suma}_{*(i,j)}$ corresponde a:

$$\begin{array}{ccccccccccc}
\mathsf{src}_{[i-2][j-2]} & + & \mathsf{src}_{[i-2][j-1]} & + & \mathsf{src}_{[i-2][j]} & + & \mathsf{src}_{[i-2][j+1]} & + & \mathsf{src}_{[i-2][j+2]} & + \\
\mathsf{src}_{[i-1][j-2]} & + & \mathsf{src}_{[i-1][j-1]} & + & \mathsf{src}_{[i-1][j]} & + & \mathsf{src}_{[i-1][j+1]} & + & \mathsf{src}_{[i-1][j+2]} & + \\
\mathsf{src}_{[i][j-2]} & + & \mathsf{src}_{[i][j-1]} & + & \mathsf{src}_{[i][j]} & + & \mathsf{src}_{[i][j+1]} & + & \mathsf{src}_{[i][j+2]} & + \\
\mathsf{src}_{[i+1][j-2]} & + & \mathsf{src}_{[i+1][j-1]} & + & \mathsf{src}_{[i+1][j]} & + & \mathsf{src}_{[i+1][j+1]} & + & \mathsf{src}_{[i+1][j+2]} & + \\
\mathsf{src}_{[i+2][j-2]} & + & \mathsf{src}_{[i+2][j-1]} & + & \mathsf{src}_{[i+2][j]} & + & \mathsf{src}_{[i+2][j+1]} & + & \mathsf{src}_{[i+2][j+2]} &
\end{array}$$

Para evitar errores de redondeo, la división debe ser la última operación en realizarse. Notar que la operación mín $(x, 255)$ se puede pensar como que x satura en 255, que es el máximo número que cabe en 8 bits. Mientras que máx $(x, 0)$ es equivalente para saturar en 0 si el resultado era negativo.

Además, dado que en los bordes no es posible calcular *ldr* por la ausencia de vecinos, deberá escribirse en esos casos el valor original del pixel. Es decir,

$$\mathsf{dst}_{(i,j)} = \mathsf{src}_{(i,j)} \text{ si } i < 2 \vee j < 2 \vee i + 2 \leq \mathsf{tamy} \vee j + 2 \leq \mathsf{tamx}$$

(con i indexado a partir de 0)

3. Enunciado

Este trabajo tiene dos objetivos principales

- Explorar el modelo de programación **SIMD**
- Realizar un análisis riguroso de los resultados de performance del procesador al hacer uso de las intrucciones **SSE**.

Para esto, cada uno de los filtros deberá ser implementado en dos versiones: una en lenguaje **C**, y una en ASM haciendo uso de las instrucciones **SSE**.

Los ejercicios que se enumeran a continuación sirven como guía mínima para que realicen optimizaciones y analicen los resultados de las mismas. También pueden plantear otras optimizaciones que surjan del desarrollo de cada filtro y acompañarlas de un respectivo análisis.

Nota: No intentar realizar el código ASM directamente, implementar primero el código C para asegurarse haber comprendido los detalles de implementación de cada filtro.

Preámbulo

A la hora de analizar la velocidad con la que se ejecuta un programa es importante tener en cuenta cuales son las limitaciones inherentes a nuestro modelo de cómputo. En una arquitectura de Von Neumann (como la de Intel), las limitaciones de performance suelen dividirse en dos grandes áreas: capacidad de cómputo y ancho de banda. Es decir, dado un hardware fijo, lo que hace que un programa no termine más rápido puede ser,

- o bien que tarda mucho tiempo en procesar cada dato
- o bien que tarda mucho tiempo en recibir y enviar datos desde y hacia la memoria.

Si un algoritmo realiza muchos cómputos y accede poco a memoria, tardará mas en procesar cada dato que en recibirlo de memoria, y por lo tanto estará limitado por la capacidad de cómputo. Si en cambio, realiza poco procesamiento pero con grandes cantidades de datos, tardará más en la transmisión de los datos que en su procesamiento, y por lo tanto estará limitado por el ancho de banda de la conexión con la memoria.

A lo largo de este trabajo experimentaremos cómo esto afecta la performance de las diferentes implementaciónes de los filtros.

Filtro *tiles*

Programar el filtro *tiles* en lenguaje C y luego en ASM haciendo uso de las instrucciones vectoriales (**SSE**).

Experimento 1 - análisis el código generado

Utilizar la herramienta `objdump` para verificar como el compilador de C deja ensamblado el código C. Como es el código generado, ¿cómo se manipulan las variables locales?¿le parece que ese código generado podría optimizarse?

Experimento 2 - optimizaciones del compilador

Compile el código de C con optimizaciones del compilador, por ejemplo, pasando el flag `-O1`[2]. ¿Qué optimizaciones realizó el compilador? ¿Qué otros flags de optimización brinda el compilador? ¿Para qué sirven?

Experimento 3 - secuencial vs. vectorial

Realice una medición de las diferencias de performance entre las versiones de C y ASM (el primero con -O1, -O2 y -O3).

¿Como realizó la medición?¿Cómo sabe que su medición es una buena medida?¿Cómo afecta a la medición la existencia de *outliers*[3]?¿De qué manera puede minimizar su impacto?¿Qué resultados obtiene si mientras corre los tests ejecuta otras aplicaciones que utilicen al máximo la CPU? Realizar un análisis **riguroso** de los resultados y acompañar con un gráfico que presente estas diferencias.

Experimento 4 - cpu vs. bus de memoria

Se desea conocer cual es el mayor limitante a la performance de este filtro en su versión ASM.

¿Cuál es el factor que limita la performance en este caso? En caso de que el limitante fuera la intensidad de cómputo, entonces podrían agregarse instrucciones que realicen accesos a memoria y la performance casi no debería sufrir. La inversa puede aplicarse si el limitante es la cantidad de accesos a memoria.

Realizar un experimento, agregando múltiples instrucciones de un mismo tipo y realizar un análisis del resultado. Acompañar con un gráfico.

Experimento 5 (*opcional*) - secuencial vs. vectorial (parte II)

Si vemos a los pixeles como una tira muy larga de bytes, este filtro en realidad no requiere ningún procesamiento de datos en paralelo. Esto podría significar que la velocidad del filtro de C puede aumentarse hasta casi alcanzar la del de ASM. ¿ocurre esto?

Modificar el filtro para que en vez de acceder a los bytes de a uno a la vez se accedan como tiras de 64 bits y analizar la performance.

Filtro *Popart*

Programar el filtro *Popart* en lenguaje C y en en ASM haciendo uso de las instrucciones vectoriales (**SSE**).

Experimento 1 - saltos condicionales

Se desea conocer que tanto impactan los saltos condicionales en el código del ejercicio anterior con `-O1`.

Para poder medir esto, una posibilidad es quitar las comparaciones al procesar cada pixel. Por más que la imagen resultante no sea correcta, será posible tomar una medida del impacto de los saltos condicionales. Analizar como varía la performance.

Si se le ocurren, mencionar otras posibles formas de medir el impacto de los saltos condicionales.

[2] agregando este flag a `CCFLAGS64` en el makefile

[3] en español, valor atípico: `http://es.wikipedia.org/wiki/Valor_atpico`

Experimento 2 - cpu vs. bus de memoria

¿Cuál es el factor que limita la performance en este caso?

Realizar un experimento, agregando múltiples instrucciones de un mismo tipo y realizar un análisis del resultado. Acompañar con un gráfico.

Experimento 3 - prefetch

La técnica de *prefetch* es otra forma de optimización que puede realizarse. Su sustento teórico es el siguiente:

Suponga un algoritmo que en cada iteración tarda n ciclos en obtener un dato y una cantidad similar en procesarlo. Si el algoritmo lee el dato i y luego lo procesa, desperdiciará siempre n ciclos esperando entre que el dato llega y que se comienza a procesar efectivamente. Un algoritmo más inteligente podría pedir el dato $i + 1$ al comienzo del ciclo de proceso del dato i (siempre suponiendo que el dato i pidió en la iteración $i - 1$. De esta manera, a la vez que el procesador computa todas las instrucciones de la iteración i, se estáran trayendo los datos de la siguiente iteración, y cuando esta última comience, los datos ya habrán llegado.

Estudiar esta técnica y proponer una aplicación al código del filtro en la versión ASM. Programarla y analizar el resultado. ¿Vale la pena hacer prefetching?

Experimento 3 - secuencial vs. vectorial

Analizar cuales son las diferencias de performace entre las versiones de C y ASM. Realizar gráficos que representen estas diferencias.

Filtro *Temperature*

Programar el filtro *Temperature* en lenguaje C y en en ASM haciendo uso de las instrucciones vectoriales (**SSE**).

Experimento 1

Analizar cuales son las diferencias de performace entre las versiones de C y ASM. Realizar gráficos que representen estas diferencias.

Filtro *LDR*

Programar el filtro *LDR* en lenguaje C y en ASM haciendo uso de las instrucciones **SSE**.

Experimento 1

Analizar cuales son las diferencias de performace entre las versiones de C y ASM. Realizar gráficos que representen estas diferencias.

3.1. Código

Para implementar los filtros descriptos anteriormente, tanto en C como en ASM se deberán implementar las siguientes funciones para imágenes en color (24 bits):

- *tiles_c*, *tiles_asm*,
- *popart_c*, *popart_asm*,
- *temperature_c*, *temperature_asm*
- *ldr_c*, *ldr_asm*

Los parámetros genéricos de las funciones son:

- *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 24 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, es decir, la cantidad de bytes que hay que avanzar para moverse a la misma columna de fila siguiente/anterior.

Además, la función tiles recibe los siguientes argumentos.

- *tamx*: Cantidad de pixeles de ancho del recuadro a copiar.
- *tamy*: Cantidad de pixeles de alto del recuadro a copiar.
- *offsetx*: Cantidad de pixeles de ancho a saltear de la imagen fuente.
- *offsety*: Cantidad de pixeles de alto a saltear de la imagen fuente.

Se puede asumir que $tamx \geq 16$, que $tamx + offsetx \leq cols$ y que $tamy + offsety \leq filas$

3.1.1. 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 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.
- El TP se tiene que poder ejecutar en las máquinas del laboratorio.

3.2. 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/videos (cargar, grabar, etc.) el programa hace uso de la biblioteca **OpenCV**, por lo que no se requiere implementar estas funcionalidades.

Para instalar las dependencias necesarias, en las distribuciones basadas en **Debian** basta con ejecutar:

```
$ make installopencv
```

Los archivos entregados están organizados en las siguientes carpetas:

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

El uso del programa principal es el siguiente:

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

El programa soporta imagenes y videos, aunque para el TP las pruebas de la cátedra se correrán sólo con imágenes. Es posible usar el programa con la mayoría de los formatos de imágenes y videos comunes. Las opciones que acepta son las siguientes:

- Los filtros que se pueden aplicar son:
  - **tiles**
    Parámetros: `tamx`, `tamy`, `offsetx`, `offsety`
    Ejemplo de uso: *tiles -i c lena.bmp 150 150 250 300*
  - **popart**
    Parámetros: Ejemplo de uso: *popart -i c marilyn.bmp*
  - **temperature**
    Parámetros:
    Ejemplo de uso: *temperature -i c -w ink.avi*
  - **ldr**
    Parámetros: Ejemplo de uso: *ldr -i c marilyn.bmp*
- *-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
- *-f, –frames*
  Genera frames independientes en vez de armar un archivo de video. Es utilizado para testing. Sólo para archivos de video.
- *-o, –output CARPETA*
  Genera el resultado en CARPETA. De no incluirse, el resultado se guarda en la misma carpeta que el archivo fuente
- *-v, –verbose*
  Imprime información adicional
- *-w, –video*
  Interpreta el archivo de entrada como video. En caso de no estar, se interpreta la entrada como una imagen.

Por ejemplo:

$ ./tp2 -v tiles -i asm lena.bmp 100 120 50 60

Aplica el filtro de **tiles** al archivo lena.bmp utilizando la implementación en lenguaje asm del filtro, pasándole como parámetro 100, 120, 50 y 60 como valores de tamx, tamy, offsetx y offsety respectivamente.

3.2.1. Tests

Para verificar el correcto funcionamiento de los filtros, se provee un comparador de imágenes, el binario de la solución de la cátedra y varios scripts de test. El binario de la cátedra se encuentra en la carpeta *solucion/bin*. El comparador de imágenes se ubica en la carpeta *solucion/tools*, y debe compilarse antes de correr los scripts (correr *make* en la carpeta *solucion/tools*).

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 *solucion/tests/data*), y las deposita en *imagenes_a_testear*. Para que este script ande se requiere la utilidad *convert* que se encuentra en la biblioteca *imagemagick*, para instalar *sudo apt-get install imagemagick*.

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.

3.2.2. Mediciones de tiempo

Utilizando la instrucción de assembly `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 a su valor luego de la llamada, podemos obtener la duración en ciclos de esa ejecución.

Las macros para medir tiempo se encuentran en el archivo `tiempo.h`. Para usarlas, se debe determinar como y donde implementarlas para poder obtener las mediciones más exactas de tiempo con granularidad de ciclo de clock.

Esta cantidad de ciclos no es siempre igual entre invocaciones de la función, ya que este registro es global del procesador y, si el programa es interrumpido por el *scheduler* para realizar un cambio de contexto, contaremos muchos más ciclos que si la función se ejecutara sin interrupciones. 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*.

3.3. 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, respondiendo *en profundidad* cada una de las preguntas de los *Experimentos*.

   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. 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).

   Las preguntas en cada ejercicio son una guía para la confección de los resultados obtenidos. Al responder estas preguntas, se deberán analizar y comparar las implementaciones de cada funciones en su versión `C` y `ASM`, mostrando los resultados obtenidos a través de tablas y gráficos.

   También se deberá comentar acerca de los resultados obtenidos. En el caso de que sucediera que la versión en C anduviese más rápidamente que su versión ASM, **justificar fuertemente** a qué se debe esto.

d) **Conclusión** Reflexión final sobre los alcances del trabajo práctico, la programación en el modelo **SIMD** 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.

4. 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 solo los archivos que tienen como nombre las funciones a implementar.

Es codicion necesaria para la aprobación de este trabajo que pasen correctamente todos los tests de la catedra.

La fecha de entrega última de este trabajo es **Jueves 8 de Mayo** y deberá ser entregado a través de la página web. El sistema solo 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**.