Organización del Computador II

Segundo Cuatrimestre de 2009

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Practico N°2

Grupo RET

Integrante	LU	Correo electrónico
Gonzales Courtois Matias	453/07	curtu_infinito73@hotmail.com
Mancuso Emiliano	597/07	emiliano.mancuso@gmail.com
Mataloni Alejandro	706/07	amataloni@gmail.com

Reservado para la catedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

Índice **2**

1. Enunciado **3**

2. Introducción **4**

3. Implementación de los algoritmos **5**

4. Imagenes procesadas **6**

5. RET-SIMD vs. OpenCV vs. RET **7**

6. Conclusión **8**

7. Manual de usuario **9**

1. Enunciado

Optimizar las funciones de procesamiento de imágenes desarrolladas en el primer trabajo práctico. Para eso se debe utilizar el modelo de programación SIMD (Single Instruction Multiple Data) y el set de instrucciones SSE de la arquitectura IA-32 de Intel. Además de los operadores de derivación ya vistos (Roberts, Prewitt y Sobel) para esta nueva entrega se pide implementar también el filtro de Frei-Chen (ver Introducción Teórica del TP1) que requiere un procesamiento en punto flotante.

Al igual que en la entrega anterior, se debe escribir la parte de interacción con el usuario y manejo de archivos en lenguaje C, utilizando la librería OpenCV. Las funciones de procesamiento de imágenes deben estar escritas en lenguaje ensamblador optimizado para instrucciones SIMD. El programa debe recibir en la línea de comandos el nombre del archivo de entrada y la operación:

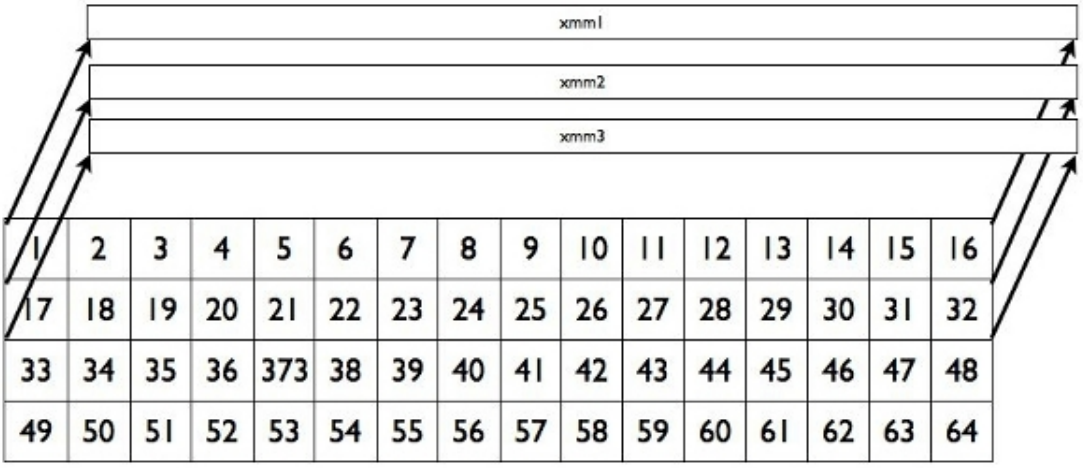
- r1 = realzar los bordes con el operador de Roberts.
- r2 = realzar los bordes con el operador de Prewitt.
- r3 = realzar los bordes con el operador de Sobel, derivación sólo en x.
- r4 = realzar los bordes con el operador de Sobel, derivación sólo en y.
- r5 = realzar los bordes con el operador de Sobel, derivación en x y en y.
- r6 = realzar los bordes con el operador de Frei-Chen.

2. Introducción

En el trabajo práctico anterior habíamos resuelto los algoritmos de forma tal, que no utilizamos las matrices de los operadores debido a que conocíamos los valores. Partimos desde este punto para optimizar el procesamiento de imágenes con instrucciones SIMD.

Pudimos desarrollar los algoritmos facilmente, a excepción de Frei-Chen, que no nos alcanzaban los registros.

Queríamos optimizar los accesos a memoria, por lo tanto levantabamos las 3 lineas en 3 registros distintos, y las manteniamos durante todo el ciclo.



Para poder operar entre los pixeles, los desempaquetabamos hasta llegar a enteros de 32 bits, y asi poder multiplicar por $\sqrt{2}$. Luego, volviamos a empaquetar a 8 bits para guardar el resultado. Esto nos consumia todos los registros SSE, y solo habíamos realizado el operador en X. Otra desventaja que traía es que podíamos procesar de a 6 pixeles, y era dificil de seguir e implementar.

La $\sqrt{2}$ siempre la manteníamos en un registro **xmm0** y asi, calcularla solo una vez. Al no alcanzarnos los registros de SSE, optamos por utilizar variables auxiliares donde contenerla, total el acceso a memoria ocurriría una sola vez, gracias a la memoria cache.

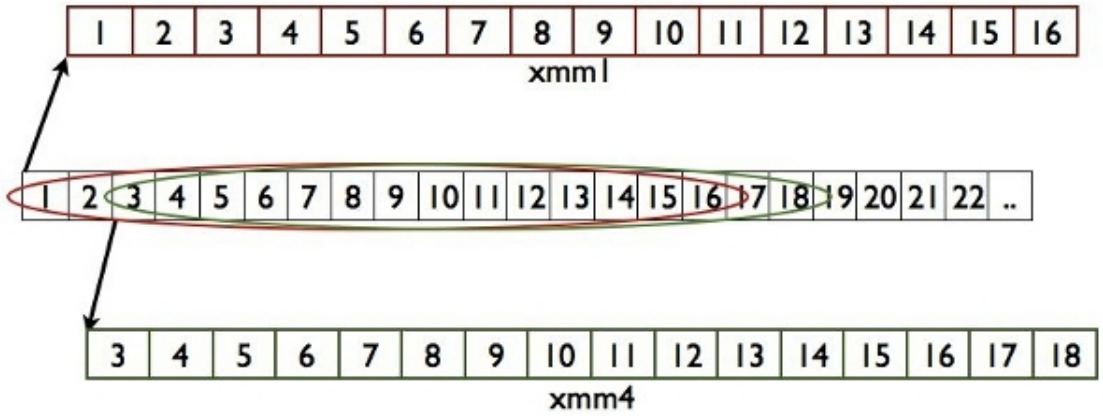
Sin embargo, la implementación del algoritmo se nos seguía complicando, sobretodo al intentar operar X e Y al mismo tiempo sin pasar por memoria, que es lo óptimo y a lo que intentabamos llegar.

Con este problema fuimos a consultar, y nos aconsejaron que tomemos el problema con otro punto de vista. En lugar de pensar el problema como sumas horizontales, y mantener los resultados con *shift* y mascaras, deberíamos pensar el problema como si se tratase de sumas verticales.

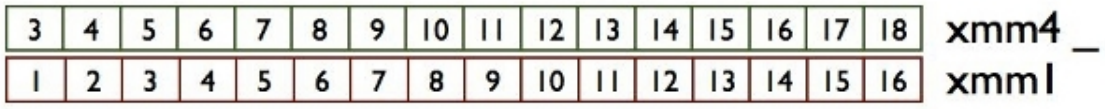
Así es como lo repensamos e hicimos la nueva implementación, mucho más clara, m'as eficiente, y con registros SSE suficientes !

3. Implementación de los algoritmos

En lugar de levantar las 3 lineas y trabajar con ellas, vamos trabajando de a una linea.



Una vez que cargamos la linea en xmm1 y la misma linea pero desplazado 2 pixeles a la derecha en xmm4, hacemos la resta de forma vertical y la mantenemos en xmm1



Procedemos de la misma manera para la segunda linea, quedando en xmm2 y para la tercer linea, en xmm3.

Ahora que tenemos procesadas cada una de las lineas, las tenemos que sumar entre si, pero sin olvidarnos de que la linea 2 (xmm2) debe ser multiplicado por $\sqrt{(2)}$. Para ello, desempaquetamos de a 4 números hasta tener enteros de 32 bits, los convertimos en números de punto flotante, los multiplicamos por $\sqrt{2}$ y luego los convertimos en enteros, y volvemos a empaquetar hasta obtenerlos en byte.

```
movdqu xmm4, xmm2
punpcklbw xmm4, xmm5   ; extendemos a 16 bits los 8 numeros de la parte baja.
punpcklwd xmm4,xmm5    ; extendemos a 32 bits los 4 numeros de la parte baja.

cvtdq2ps xmm4,xmm4     ; convertimos a float
mulps xmm4,xmm0        ; multiplicamos por raiz(2)
cvtps2dq xmm4, xmm4    ; lo volvemos a convertir en enteros de 32

packssdw xmm4,xmm4     ; xmm4 = primeros 4 resultados
packuswb xmm4,xmm4     ; los devolvemos a byte
```

Ahora xmm2 contiene los números multiplicados por $\sqrt{2}$ listos para ser sumados con los otros.

Por último, sumamos con saturación los tres registros de forma vertical para obtener los resultados y lo asignamos a **xmm7**.

Esto es válido dado que:

$$\begin{array}{ccc|ccc} A & B & C & -1 & 0 & 1 \\ D & E & F & -\sqrt{2} & 0 & \sqrt{2} \\ G & H & I & -1 & 0 & 1 \end{array}$$

$$\begin{aligned} &\Rightarrow \text{ - A - D}\sqrt{2}\text{ - G + C + F}\sqrt{2}\text{ + I} \\ &\Leftrightarrow \text{ - A + C - D}\sqrt{2}\text{ + F}\sqrt{2}\text{ - G + I} \\ &\Leftrightarrow \text{C - A + (F - D)}\sqrt{2}\text{ + I - G} \end{aligned}$$

Acerca de los bordes de la imágen, hay dos soluciones

- Agregar un pixel demás en los bordes, ya sea clonandolos o en negro, para hacer posible el calculo de la imágen por completo.
- Ignorar los bordes y dejarlos en negro

Nosotros optamos por ignorar los bordes y dejarlos en negro.

Hasta acá, en **xmm7** tenemos el resultado de haber procesado la imágen con el operador Frei-Chen pero solo en X.

Para calcular Frei-Chen en Y, el algoritmo es muy similar, en lugar de tomar la primer fila y su corrimiento de dos pixeles, tomamos la primer fila y la tercera, pues en Y la segunda fila la ignoramos por ser 0.

Despues se procede de la misma manera.

4. Imagenes procesadas

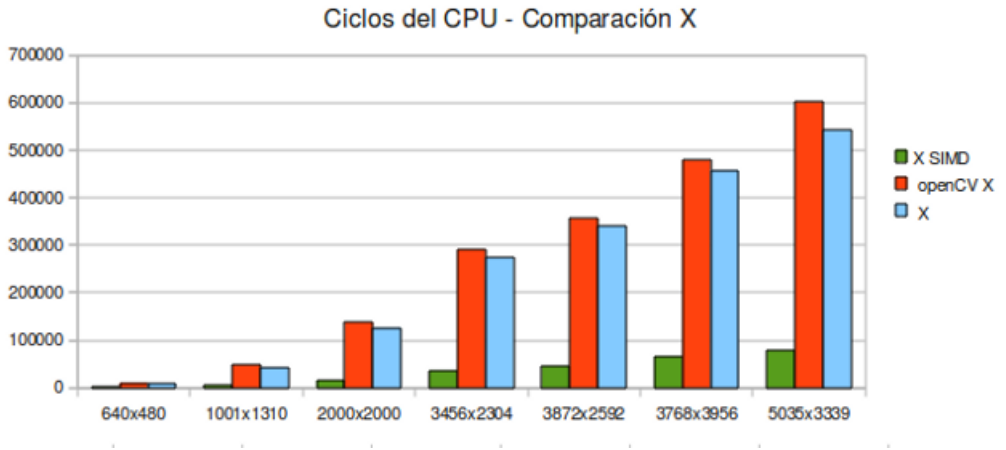
Las imágenes procesadas se encuentran en el directorio `resultados` debido a que el procesamiento de las imágenes no se puede apreciar en una impresión. Hay que ejecutar el programa, para elegir el operador a utilizar y luego alguna de las imágenes del directorio `src/images/`.

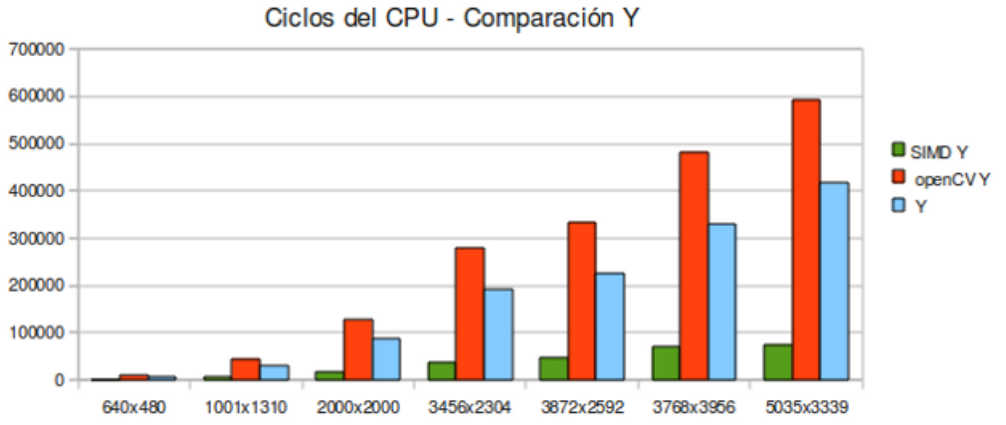
5. RET-SIMD vs. OpenCV vs. RET

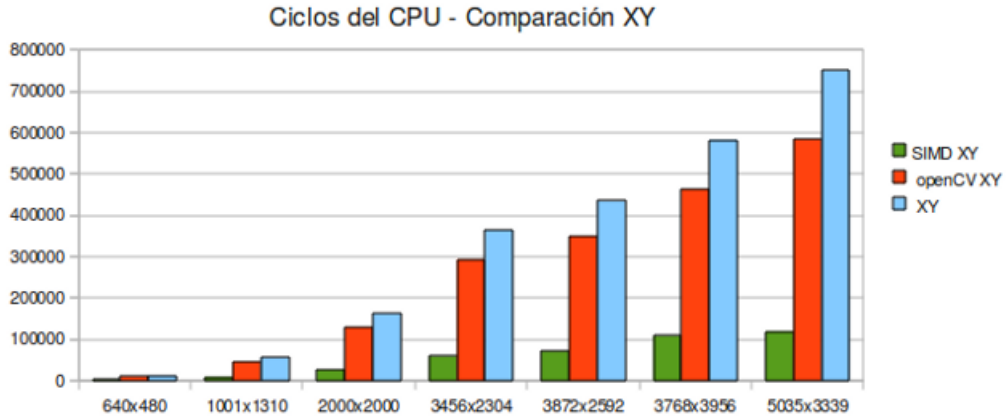
En esta sección podemos ver la diferencia entre procesar las imágenes con registros de proposito general, y con registros MMX. Entonces, utilizamos los algoritmos desarrollados en el Trabajo Práctico 1 y los comparamos con los recién explicados.

Además, aprovechamos y mostramos al mismo tiempo la comparación con la implementación de la librería `OpenCV`.

De esta forma, podemos ver las 3 implementaciones distintas, agrupadas por operador (X, Y o XY). La librería OpenCV implementa su propio algoritmo para el operador de Sobel con registros de próposito general. En cambio nosotros, a diferencia de OpenCV, implementamos el algoritmo con instrucciones SIMD. Para comparar los algoritmos utilizamos imágenes de distintos tamaños, y medimos los ciclos del CPU para comparar la eficiencia de las distintas implementaciones.







Para realizar estos gráficos, seleccionamos distintas imágenes con distinta resolución, y corrimos cada algoritmo un total de 20 veces para luego calcular un promedio. Para ajustar la escala del gráfico, la cantidad de ciclos esta representada en miles, pues sino se hacía despreciable respecto a las imágenes más pequeñas.

6. Conclusión

Gracias a estos gráficos podemos contrastar la velocidad que nos brindan las instrucciones `SIMD`. También llegamos a la conclusión de que la librería `OpenCV` no utiliza instrucciones `SIMD`, y esto se debe a que la librería se utiliza en muchos lugares y no todos los procesadores implementan esta tecnología. Si supieramos que estos algoritmos correrían sobre procesadores IA-32 compatibles con instrucciones SIMD, definitivamente sería la mejor opción. Pero, dado que no son los únicos procesadores, la librería prioriza la portabilidad de los algoritmos frente a la optimalidad. Por eso, tenemos una diferencia abismal entre las distintas implementaciones.

7. Manual de usuario

1. Ejecutar el comando `make` o `make install` desde el directorio `tp2`.
2. Acceder al directorio `exe` y ejecutarlo
   a) Directo `./tp r1 images/lena.bmp`
   b) Interactivo `./tp`
3. Las imágenes procesadas se encuentran en el directorio `resultados`.
4. En el directorio `src/images/` se encuentran algunas imágenes de ejemplo para procesar.
5. Para agregar imágenes a procesar, estas deben ser agregadas al directorio `src/images/`.

Archivos

```
tp2/
|-- Makefile
|-- enunciado
|   `-- Enunciado.pdf
|-- exe
|-- informe
|   `-- Informe.pdf
|-- resultados
`-- src
    |-- Makefile
    |-- asmFrei_Chen.asm
    |-- asmPrewitt.asm
    |-- asmRoberts.asm
    |-- asmSobelXY.asm
    |-- images
    |   |-- 10mb.jpg
    |   |-- 1mb.jpg
    |   |-- 2mb.jpg
    |   |-- 5mb.jpg
    |   |-- emma.jpeg
    |   |-- foto3.jpg
    |   |-- lena.bmp
    |   `-- pink.jpg
    `-- main.c
```