UNIVERSIDAD DE BUENOS AIRES

Facultad de Ciencias Exactas y Naturales
Departamento de Computación
Organización del Computador

TRABAJO PRÁCTICO NÚMERO 2

Nombre del Grupo: Napolitana con Jamón y Morrones

Alumnos:

Izcovich, Sabrina | sizcovich@gmail.com | LU 550/11

López Veluscek, Matías | milopezv@gmail.com | 926/10

Tito, Matías Gonzalo | matias.tito@gmail.com | 437/06

Índice

1. Introducción

El objetivo de este trabajo práctico fue experimentar utilizando el modelo de programación SIMD. Para ello, fue requerido implementar seis filtros para procesamiento de imágenes (Recortar, Halftone, Umbralizar, Colorizar, Efecto Plasma y Rotar) tanto en *C* como en *Assembler*.
Por otro lado, debimos analizar la performance de un procesador al hacer uso de las operaciones SIMD. Para ello, realizamos comparaciones de velocidad entre los dos tipos de implementaciones realizados utilizando la herramienta Time Stamp Counter (TSC) del procesador.

2. Desarrollo

hay que decir que en rotar.c si definimos a la raiz como un double pierde precision contra ponerlo como float.

hay que poner el pseudocodigo detallado que utilizamos explicando por que recorrimos de la manera que recorrimos la matriz y por que nos parecio lo mas efectivo y eficiente para programar, detallando para que sirve cada cosa. Tambien tenemos que decir que cosas hicimos para facilitar la implementacion en asm, o sea si lo hicimos pensando en como ibamos a resolverlo en asm o si directamente lo hicimos de una manera que fuera rapida. algo que se me ocurre es que preferimos, por ejemplo, usar muchas variables para definir pequeñas operaciones que usaba una sola ecuacion en vez de que fuera todo resuelto de una para poder darles tipos a esas operaciones y que el resultado fuera mas preciso y acertado a lo pedido.

tenemos que decir que tipos de variaciones obtuvimos al cambiar el codigo en detalles, por ejemplo, la ejecucion del codigo resulto un poco mas rapida al poner ++j que j++. tambien cambiaba el resultado si iterabamos primero columnas y despues filas o viceversa (esto es cierto???).

3. Resultados

tenemos que ver el codigo en asm creado por el .c para decir por que nuestro asm funciona mas rapido que el generado por el .c, generalmente produce mas ciclos innecesarios o quizas contempla algun caso que no es importante (por ejemplo una altura o un ancho negativo). bueno obviamente hay que concluir que el .asm es mucho mas rapido que el .c.
tenemos que hacer un grafico, creo que podriamos hacer uno que para cada funcion

que hicimos ponga el tiempo que tardo en asm y en .c y que una por un lado todos los .c y por el otro todos los .asm cuestion de ver precisamente cual es la relacion entre la velocidad de un asm y de un .c. tambien tenemos que hacer uno que muestre graficamente el contenido de los xmm a medida que avanza la ejecucion del programa aca hay que poner tablas y graficos con los resultados. analizar y comparar las implementaciones. podriamos ejecutar el timing en distintas computadoras y ver que cambia respecto del procesador usado.

4. Conclusión

Reflexion final sobre alcance de la programacion vectorial a bajo nivel.