Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 2
2^o cuatrimestre de 2014

`$Date: 2014/10/05 18:17:07 $`

1. Objetivos

Ejercitar algunas de las técnicas de optimización de software vistas en el curso. Familiarizarse con elementos de assembler i386 y programación SIMD (single instruction, multiple data); y con el uso de herramientas de análisis, medición y profiling de sistemas de cómputo.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la sección 5, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Descripción

Se trata de un modificar un programa de tal forma que dibuje el conjunto Multibrot de orden 3 [1] [2] [3] y sus vecindades, en el cual la lógica de de cómputo del fractal deberá tener soporte nativo para procesadores x86.

El código fuente con la versión inicial del programa, se encuentra disponible en [4]. El mismo deberá ser considerado como punto de partida de todas las implementaciones. En particular notar que este código implementa polinomios de orden 2, para lo cual cada grupo deberá realizar las modificaciones a la lógica de cómputo del programa descriptas en las sección 4.4.

4.1. Programa

El programa recibe, por línea de comando, la descripción de la región del plano complejo y las características del archivo imagen a generar. No interactúa con el usuario, ya que no se trata de un programa interactivo, sino más bien de una herramienta de procesamiento *batch*. Al finalizar la ejecución, y volver al sistema operativo, el programa habrá dibujado el fractal en el archivo de salida.

El formato gráfico a usar es PGM o *portable gray map* [5], un formato simple para describir imágenes digitales monocromáticas.

4.2. Algoritmo

.

El algoritmo básico es simple: para algunos puntos c de la región del plano que estamos procesando haremos un cálculo repetitivo. Terminado el cálculo, asignamos el nivel de intensidad del pixel en base a la condición de corte de ese cálculo.

El color de cada punto representa la "velocidad de escape" asociada con ese número complejo: blanco para aquellos puntos que pertenecen al conjunto (y por ende la "cuenta" permanece acotada), y tonos gradualmente más oscuros para los puntos divergentes, que no pertenezcan al conjunto.

Más específicamente: para cada pixel de la pantalla, tomaremos su punto medio, expresado en coordenadas complejas, $c = c_{re} + c_{im}i$. A continuación, iteramos sobre $f_{i+1}(c) = f_i(c)^3 + c$, con $f_0(c) = c$. Cortamos la iteración cuando $|f_i(c)| > 2$, o después de `N` iteraciones.

En pseudo código:

```
para cada pixel $p {
        $f = $c = complejo asociado a $p;
        for ($i = 0; $i < $N - 1; ++$i) {
                if (abs($f) > 2)
                        break;
                $f = $f * $f * $f + $c;
        }
        dibujar el punto p con brillo $i;
}
```

Así tendremos, al finalizar, una representación visual de la cantidad de ciclos de cómputo realizados hasta alcanzar la condición de escape (ver figura 1).

4.3. Interfaz

A fin de facilitar el intercambio de código *ad-hoc*, normalizaremos algunas de las opciones que deberán ser provistas por el programa:

- `-r`, o `--resolution`, permite cambiar la resolución de la imagen generada. El valor por defecto será de 640x480 puntos.
- `-c`, o `--center`, para especificar el punto central de la porción del plano complejo dibujada, expresado en forma binómica (i.e. $a + bi$). Por defecto usaremos `-0.60 + 0.25i`.
- `-w`, o `--width`, especifica el ancho del rectángulo que contiene la región del plano complejo que estamos por dibujar. Valor por defecto: `0.1`.

- `-H`, o `--height` sirve, en forma similar, para especificar el alto del rectángulo a dibujar. Valor por defecto: `0.1`.
- `-o`, o `--output`, coloca la imagen de salida, (en formato PGM [5]) en el archivo pasado como argumento; o por salida estándar `-stdout-` si el argumento es "-".
- `-m`, o `--method`, permite seleccionar dinámicamente la implementación a usar para generar la imagen de salida: `generic` para seleccionar la implementación genérica, en lenguaje C; y `sse` para usar, cuando esté disponible, la implementación con soporte para hardware SSE. Por defecto, la implementación de referencia utiliza `generic`.
- `-n`, o `--nthreads`, permite fijar la cantiad de threads que intervienen en el cómputo del fractal. El valor por defecto de la implementación de referencia es `1`, es decir, un único hilo de cómputo.

4.4. Entregables

El entregable producido en este trabajo deberá implementar la lógica de cómputo del fractal (Multibrot de orden 3) en assembly SSE, es decir deberá tener soporte nativo para procesadores intel.

Para ello, cada grupo deberá tomar el código fuente de base para este TP, [4], y reescribir la función `sse_plot()` de tal forma de que permita generar un Multibrot de orden 3 (la implementación provista implementa polinomios de grado 2). Esta función está ubicada en el archivo `sse_plot.c`.

Asimismo, cada grupo deberá proponer e implementar un esquema alternativo para el modelo de cómputo paralelo.

Por otro lado, si bien este trabajo no tiene requerimientos específicos de *speedup*, nos interesa conocer el efecto que tienen estos cambios en relación a la versión de base *single-threaded* codificada en C (opción `generic`).

Para ello, cada grupo deberá realizar las mediciones que considere necesarias, fundamentándolas en el TP.

4.5. Casos de prueba

El informe trabajo práctico deberá incluir una sección dedicada a verificar el funcionamiento del código implementado. Para ello, será necesario escribir pruebas orientadas a probar el programa completo, ejercitando los casos más comunes de funcionamiento, los casos de borde, y también casos de error.

4.6. Ejemplos

Generamos un dibujo usando los valores por defecto, barriendo la región rectangular del plano comprendida entre los vértices `-0.65 + 0.30i` y `-0.55 + 0.20i`.

```
$ tp1-2014-2-bin -o uno.pgm
```

La figura 1 muestra la imagen `uno.pgm`.

A continuación, hacemos *zoom* sobre la región centrada en `-0.6 + 0.6i`, usando un rectángulo de `0.05` unidades de lado. El resultado podemos observarlo en la figura 2.

```
$ tp1-2014-2-bin --center -0.6+0.6i --width 0.05 --height 0.05 -o dos.pgm
```

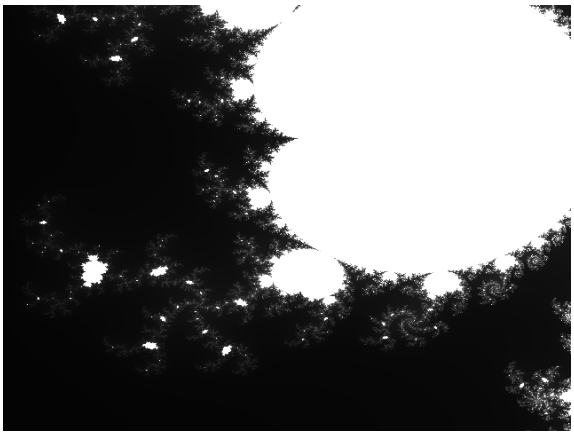

Figura 1: Región barrida por defecto.

Figura 2: Región comprendida entre `-0.625 + 0.625i` y `-0.575 + 0.575i`.

5. Informe

El informe deberá incluir:

- Documentación relevante al diseño e implementación del programa.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente.
- Indicadores de performance de la aplicación, que permitan cuantificar el efecto de los cambios introducidos tomando como base de comparación la implementación *single-threaded* en lenguaje C.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, en lenguaje C.
- Este enunciado.

6. Fecha de entrega

La última fecha de entrega y presentación sería el martes 25/11.

Referencias

[1] `http://en.wikipedia.org/wiki/Mandelbrot_set` (Wikipedia).

[2] Introduction to the Mandelbrot Set.
`http://www.olympus.net/personal/dewey/mandelbrot.html`.

[3] Smooth shading for the Mandelbrot exterior.
`http://linas.org/art-gallery/escape/smooth.html`. Linas Vepstas. October, 1997.

[4] Código fuente con el esqueleto del trabajo práctico.
`http://www.fiuba7504.com.ar/tp2-2014-2-src.tar.gz`.

[5] PGM format specification.
`http://netpbm.sourceforge.net/doc/pgm.html`.