TALLER DE PROGRAMACIÓN PARALELA
LABORATORIO 1

Conjuntos Mandelbrot bajo procesos de sistema LINUX y Memoria compartida POSIX

Autor: Mario López, Profesor: Fernando Rannou, Ayudante: Danilo Aburto
UNIVERSIDAD DE SANTIAGO DE CHILE – DEPARTAMENTO DE INFORMÁTICA
Santiago, 21 de Octubre de 2013

MOTIVACIÓN

Los conjuntos fractales siempre han sido de grán interés para los matemáticos y científicos en general, principalmente debido a su presencia en complejas estructuras de la naturaleza. Estos conjuntos caracterizan por su autosimilitud o invariabilidad con relación a la escala o zoom que se aplique sobre sus partes. A continuación se da cuenta de un programa que explota la concurrencia de la computación para generar conjuntos Mandelbrot, uno de los conjuntos fractales mas conocidos y estudiados.

OBJETIVO

El objetivo es construir un programa en C capáz de hacer uso de la concurrencia y eventual paralelismo de un computador, para generar una imagen de un fractal Mandelbrot. A partir de lo anterior se desea estudiar cómo se observa un *speedup* en los tiempos de respuesta del programa en función del número de procesadores que se encuentren trabajando.

ALGORITMO

Sea *c* un número complejo cualquiera. A partir de *c*, se construye una sucesión por recursión de la siguiente forma:

$$\begin{cases} z_0 &= 0 & \text{(término inicial)} \\ z_{n+1} &= z_n^2 + c & \text{(relación de inducción)} \end{cases}$$

Si esta sucesión queda acotada, se dice que el número *c* pertenece al conjunto Mandelbrot, y si no, queda fuera de éste. (*Vease* http://es.wikipedia.org/wiki/Conjunto_de_Mandelbrot).

CONCURRENCIA

El procedimiento de separación de las tareas para explotar la capacidad de cada procesador, ocurre en función de cada número *c* (correspondiente a un pixel de la imagen) para ser tratado de forma independiente por alguna unidad de procesamiento. De esta forma el espacio o región sobre la cual se desea realizar el cálculo es mapeada y transformada a un espacio discreto en forma de matriz de dos dimensiones. Esta discretización del espacio, se realiza a partir de un *step* o ancho de cada grano a muestrear. Un ejemplo de esto es que a partir del espacio [-1, 1] en el eje real, y [-1, 1] en el eje complejo, al realizar un muestreo de 0.1, entonces se genera una matriz de 21 filas y 21 columnas, las cuales generan 441 pixeles que representan números *c*, sobre los cuales aplicar el algoritmo Mandelbrot.

A partir de lo anterior, es que se ha diseñado un programa que toma estos 441 *jobs* o tareas, y los distribuye entre los procesos de acuerdo a lo siguiente: Cada proceso es numerado con un identificador (*pid*) y además conoce cuantos procesos estan procesando al igual que él (*warpsize*). Entonces, los procesos comienzan a trabajar la matriz, procesando una fila a la vez y de forma concurrente respecto a sus pares. Cada proceso comienza tomando la fila *pid-ésima* y despues de procesarla, avanza *warpsize* hasta la fila *pid* + *warpsize*. De forma sucesiva los procesos realizan saltos de tamaño *warpsize* hasta que han detectado que se han salido del espacio, o dicho de otra forma, realizan esto siempre y cuando los saltos sean menores al total de filas de la matriz.

RESULTADOS

La distribución descrita en la sección anterior fue probada sobre una máquina de 24 procesadores y un problema Mandelbrot de 6.151 filas y 6.151 columnas, con un total de 37.834.801 *jobs*. El cual representa la discretización de el espacio complejo [-0.748766713922161, 0.123640844894862] en el eje real, y [-0.748766707771757, 0.123640851045266] en el eje complejo, y bajo una tasa de muestreo de 0.000000000001. Lo que finalmente resulta en la siguiente imagen.

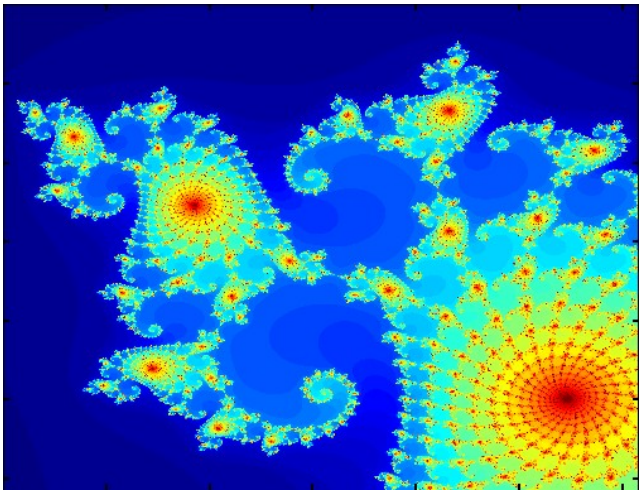

FIGURA 1. Mandelbrot de pruebas.

El resultado final, es decir, la imagen, fue generada en un total de 15 segundos con 24 procesos sobre 24 procesadores. Por otro lado, se realizó la prueba de ejecutar el programa con un proceso para contrastar lo anterior con el tiempo secuencial, el cual fue de 140 segundos. Por lo tanto, el *speedup* observado como máximo en esta máquina fue de 9,3X.

A partir de lo anterior se ejecutó el programa secuencial y luego se incrementó el número de procesos para observar cómo crecía el speedup del algoritmo. En la Figura 2, se observa que en la medida que se incremente la cantidad de procesos, disminuye el tiempo similar a una curva logaritmica, pero es importante notar que a partir de los 10 procesos no se observa una mejora importante en el tiempo de respuesta, sino mas bien se estabiliza.

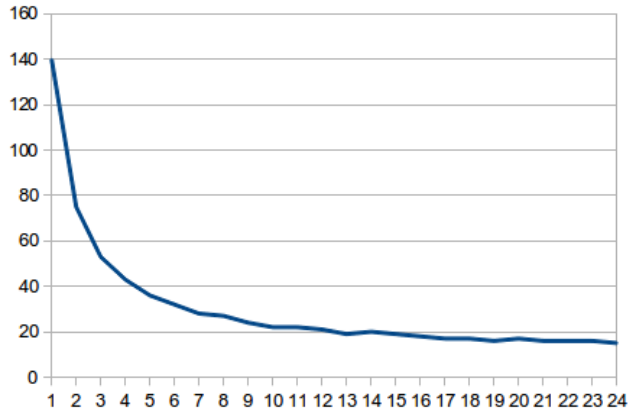

FIGURA 2. Tiempo de respuesta en función al número de procesos.

Por otro lado, en la Figura 3 se observa cómo el *speedup* de cada una de las ejecuciones crece de forma sostenida, pero a partir los 10 procesos, la curva comienza a presentar valores inesperados y azarosos.

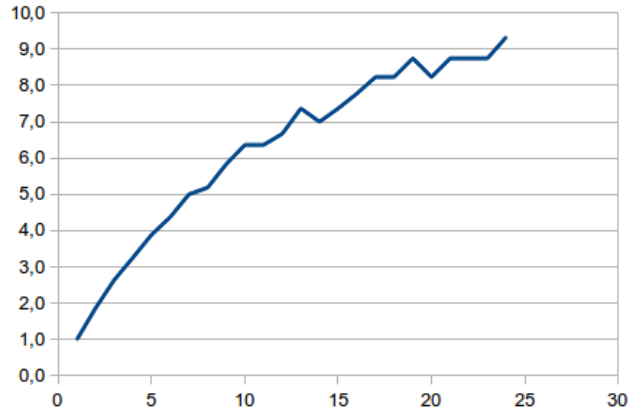

FIGURA 3. Speedup en función al número de procesos.

CONCLUSIONES

En general, el programa no presenta un *speedup* de alta calidad, ante un problema que no tiene dependencias entre los *jobs* y que por lo tanto no es necesario realizar ninguna sincronización entre los procesos. Se presume que los tiempos de ejecución disminuirían si en vez de utilizar procesos concurrentes, el trabajo se realizara con hebras, cuyo costo en memoria es menor y no presentarían una carga alta en *overhead.*