Javier Falcón 2016-5265
Manuel Molina 2016-5468

Introducción

Anteriormente en clase se llevaron a cabo experimentos que nos permitieron entender y observar el comportamiento de computaciones en paralelo. En esta ocasión, nuestro objetivo es analizar un algoritmo paralelo que utiliza paso de mensajes. El ejercicio está centrado en la generación de una imagen que contenga un fractal de Mandelbrot. El particular interés en la paralelización de este algoritmo puede deberse a que la computación del color de cada píxel en la foto requiere de mucha computación, por lo que una solución secuencial puede ser bastante ineficiente. Un fractal de Mandelbrot es un conjunto de puntos en un plano complejo que nunca excederán un límite.

Para la realización de esta práctica fue debido tomar en cuenta aspectos como las tareas a paralelizar, el tamaño de dichas tareas y sus dependencias entre sí (si existen). Dicho esto, se pide la toma de, al menos, 10 medidas de tiempo en la ejecución del algoritmo, en las cuales debe variarse el número de procesos de manera creciente. Con estos resultados podremos analizar aspectos de granularidad y cómo la cantidad de comunicación puede afectar en el *performance* del algoritmo.

Javier Falcón 2016-5265
Manuel Molina 2016-5468

Resolución del problema

Con un poco más de experiencia utilizando la interfaz de MPICH2, la implementación del código que soluciona el problema fue mucho más llevadera. En nuestro caso, se implementó en primera instancia una versión secuencial del código para entender con mayor facilidad el comportamiento de el algoritmo. Mediante consultas al libro de Wilkinson, *Parallel Programming*, pudimos entender lo que se pedía para poder generar los datos del archivo. Como se estableció anteriormente, el fractal es un conjunto de número complejos que no exceden un límite y vienen generados por la siguiente fórmula

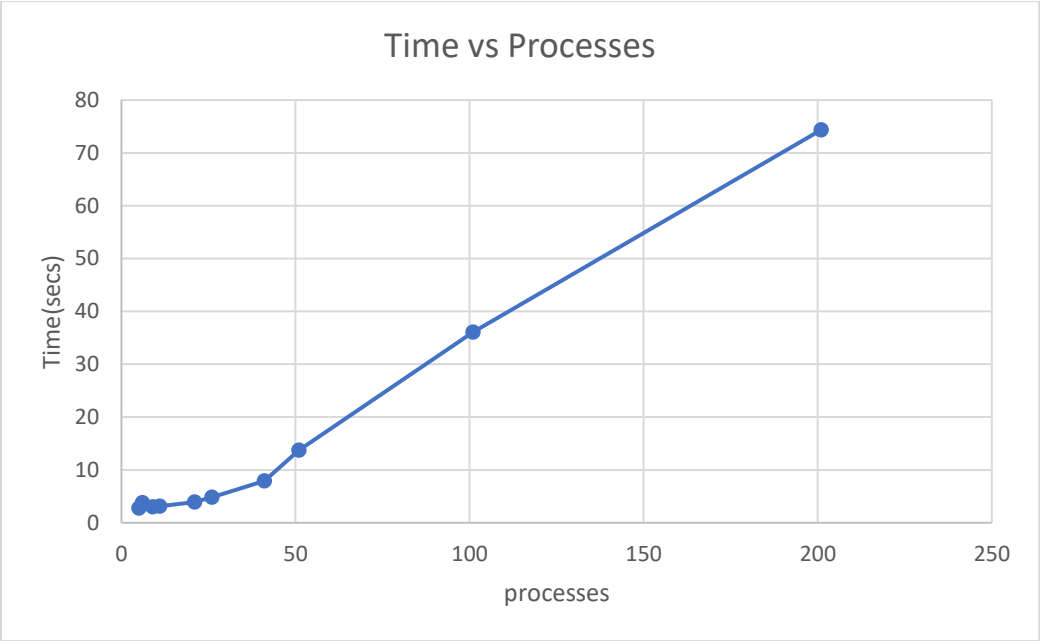
$$Z_{k+1} = Z_k^2 + C.$$

En ella, Z_{k+1} representa el número complejo Z en la $(k + 1)$ iteración. Este cálculo se detendrá cuando la magnitud de Z sea mayor que 2, lo cual indica que Z eventualmente será infinito, o cuando se llegue a un límite de iteraciones. Tomando esto en cuenta, implementamos 3 funciones para realizar los cálculos: **toReal()** y **toImaginary()** se encargarán de generar el número complejo C para que la función **calcMandelbrot()** pudiera hacer el proceso iterativo anteriormente mencionado. En este punto ya comprendíamos que la tarea a paralelizar era el cálculo de cada píxel para que la imagen pudiera generarse. Ahora bien, ¿cuál es el tamaño de tarea adecuado? Decidimos que el archivo se dividiría en 200 franjas, las cuales pueden contener varias filas de píxeles. De esta manera, cada proceso sería el responsable de calcular al menos 1 franja entera, dependiendo de cuántos procesos se inicien.

Si bien las tareas son independientes, si se necesita un orden en cuanto a la colocación de los valores de los pixeles dentro del archivo para que la imagen quede correctamente formada. Por esta razón, decidimos utilizar una matriz, cuyo tamaño está dado por la resolución del archivo que tendrá la imagen. En nuestro caso, se estableció una resolución de 5000x5000. De esta manera, cada proceso guardaría los cálculos que le fueron asignados y los guardaría en la sección de la matriz que se le especificó. Finalmente, cuando el proceso maestro recoge toda la data de sus procesos hijos, escribe en el archivo los valores de los píxeles utilizando la matriz resultante.

Javier Falcón 2016-5265
Manuel Molina 2016-5468

Análisis de los datos



Gráfica 1 Comparación de tiempo de ejecución y cantidad de procesos

La **Gráfica 1** nos ilustra los resultados obtenidos del experimento. Para la generación de los datos realizamos lo siguiente: se dejó la cantidad de franjas fijadas en 200. Como la resolución escogida fue de 5000x5000, cada franja conllevaría 25 líneas o filas del archivo. Dicho esto, se tomaron 10 muestras de tiempo con cantidades de procesos diferentes y de manera creciente, siempre tomando los divisores exactos de 200 para que la repartición quedara pareja.

Como se puede observar, la gráfica crece a medida que la cantidad de procesos aumenta. Este comportamiento debe estar relacionado con la granularidad y la comunicación entre procesos: mientras la cantidad de procesos aumenta, el tamaño de tarea correspondiente a cada uno se vuelve más pequeño, obteniendo una granularidad más fina. En clase discutimos que mientras la granularidad sea más fina, más comunicación entre procesos tendremos. Se puede apreciar cómo afecta el *performance* del programa la cantidad de comunicación: mientras más comunicación tengamos, más tiempo toma el

programa para computar una salida ya que se gastará más tiempo enviando y recibiendo mensajes que computando lo que se necesita.

Conclusiones

En resumen, a parte de haber practicado la generación de un fractal de Mandelbrot y a comprender su naturaleza, podemos concluir lo siguiente:

1. Confirmamos que una menor granularidad nos arroja tiempos de ejecución más altos debido a que cada la computación se limita a una mayor comunicación en lugar de a una mayor computación, lo cual indica que es importante encontrar un balance en cuanto a la cantidad de procesos que se abren para computar.
2. Aprendimos a identificar tareas que no presentan dependencia entre sí, lo cual implica que se puede lograr una "programación enteramente paralela".
3. Entendemos que es preferible asignar tareas más grandes y tener "pocos procesos".