Implementación de métodos computacionales

TC2037.2

Actividad Integradora 5.3:

Resaltador de Sintaxis Paralelo

Profesor(a):

Alejandro de Gante

Abraham Mendoza Pérez	A01274857
Luis Alonso Martínez García	A01636255
Aldo Alejandro Degollado Padilla	A01638391

Viernes 04 de junio de 2021

En esta actividad integradora lo que se nos pidió que hiciéramos, era optimizar nuestro programa ya existente (resaltador léxico) con la ayuda de hilos. De esta forma se aprovecha más el poder de nuestro procesador ya que hace varias tareas paralelamente, lo que reduce el tiempo de ejecución del programa. En las clases, el profesor nos explicó que quería que nuestro programa, al ejecutarse, creara un hilo productor que se encargara de leer un directorio, dado por el usuario, para encontrar dentro de él, todos los archivos .scm (lenguaje Scheme). Posteriormente, debíamos crear un cierto número de hilos consumidores que se encargaran de analizar uno por uno, todos los archivos encontrados por el hilo productor. Estos hilos consumidores estarán trabajando paralelamente, por lo que, en teoría, la ejecución del programa será más rápida.

Nosotros, después de crear y darle la instrucción al hilo productor, realizamos un join para esperar a que el hilo terminara la lectura del directorio, ya que, si no lo hacíamos de esta manera, los hilos consumidores no tendrían ningún archivo que analizar. Esto pasa así debido a que nosotros le damos un método a ejecutar al hilo productor que tiene como parámetro el path a analizar y que retorna una cola con toda la lista de archivos .scm. Después de finalizado el proceso del archivo productor, creamos el número de hilos que consideramos óptimos y comenzamos a darles un archivo de la cola a cada uno para que lo analicen. Cuando la cola se termina, hacemos un join a todos los hilos que todavía se encuentran analizando un archivo, para evitar algún problema en el programa o que este termine su ejecución antes de que todos los análisis hayan sido completados.

Speedup $S_p = T_1 / T_p$

1. **1 Hilo** – 21 milisegundos
2. **5 hilos** – 17 milisegundos
    a. $S_5 = 21 \text{ ms} / 17 \text{ ms} = 1.23$
3. **10 hilos** – 14 milisegundos
    a. $S_{10} = 21 \text{ ms} / 14 \text{ ms} = 1.5 \text{ ms}$
4. **20 hilos** – 18 milisegundos
    a. $S_{20} = 21 \text{ ms} / 18 \text{ ms} = 1.16 \text{ ms}$

La complejidad del algoritmo como tal que se encarga del análisis y resaltado de sintaxis sigue siendo el mismo que tuvimos en la actividad integradora anterior, ya que el concepto de hilos que implementamos para esta actividad no interviene en este, sino en poder realizar todo ese proceso múltiples veces, para hacerlo más rápida y eficientemente.

Por lo que, debido al código que tenemos en la parte del análisis, en el cual se repite varias vece un ciclo for donde leemos cada carácter, pues lo tenemos dentro de un while donde leemos cada línea del texto de entrada, en realidad el número de iteraciones totales termina siendo el número de caracteres que tenemos en total. Debido a esto, el tiempo de ejecución sería T(n) = n. 3. Por ello, si calculamos entonces la complejidad en base a nuestro tiempo de ejecución obtendremos que: { O(1) } T(n) = n => O(n), o sea que, nuestro algoritmo implementado sería de complejidad lineal.

Al haber implementado la programación paralela en nuestro resaltador léxico, logramos darnos cuenta de la gran ventaja que puede ofrecer al ejecutar el programa de manera más rápida y eficiente. Sin embargo, pueden presentarse consecuencias dependiendo del equipo en el que se ejecute el programa ya que en caso de no contar con un procesador lo suficientemente potente, el rendimiento de este se verá afectado y en casos muy extremos podría llegarse dañarse. De igual manera, debido a que el propósito de nuestro programa es el análisis de un directorio dado por el usuario, se podría llegar a pensar que incurrimos en el robo de datos de ese directorio de la computadora del usuario. Así mismo, ya que el código se encuentra público en la plataforma GitHub se podría modificar para este fin.

Para terminar, nos gustaría mencionar que durante la elaboración de esta situación problema aprendimos la utilidad de la programación paralela concurrente y la utilización hilos. Vimos en carne propia, como estas técnicas nos ayudaron a reducir el tiempo de ejecución de nuestro programa. A la vez que aprendimos cómo es que al aumentar o disminuir el número de hilos durante la ejecución

afectaba a nuestro programa. Con esta actividad concluimos nuestra participación en la materia TC2037, de la que nos llevamos un sinfín de aprendizajes.

Referencias

Hinnant, H. (2012). Get return code from std::thread? [duplicate]. *StackOverflow*. Recuperado de https://stackoverflow.com/questions/12320003/get-return-code-from-stdthread

Sharkey, K. (2018). Listing the Files in a Directory. *Microsoft Windows Developer.* Recuperado de https://docs.microsoft.com/en-us/windows/win32/fileio/listing-the-files-in-a-directory

Visual Studio Code. (S.F.). *Using GCC with MinGW*. Recuperado de https://code.visualstudio.com/docs/cpp/config-mingw

Link al vídeo explicativo:

https://youtu.be/xkS6cvZpVBI

Link al proyecto en GithHub:

https://github.com/Diplex09/TC2037/tree/main/ActividadIntegradora5.3