Trabajo Práctico 1: *pthreads*

Sistemas Operativos - Primer cuatrimestre de 2019

Fecha límite de entrega: Sábado 20 de abril a las 23:59

1. Introducción

El problema del Árbol Generador Mínimo (*Minimum Spanning Tree* en inglés) fue resuelto por primera vez en 1926 por Otakar Borůvka[1] y al día de hoy tiene numerosas aplicaciones en organización, ruteo y partición de redes.

La definición del problema es la siguiente: Dado un grafo conexo y no dirigido G con pesos en sus ejes, encontrar el árbol generador de peso total mínimo. Es decir, encontrar un subgrafo que sea árbol, que contenga a todos los vértices de G y tal que la suma de los pesos de los ejes del subgrafo sea igual o menor a la de cualquier otro árbol generador de G.

El objetivo de este trabajo es diseñar e implementar un algoritmo **paralelo** para el problema del árbol generador mínimo utilizando `pthreads`. Se espera que ejerciten —en el contexto de un problema real— algunos de los temas de sincronización vistos en la materia: contención, concurrencia, condiciones de carrera, secciones críticas, instrucciones atómicas, semáforos, memoria compartida, entre otros problemas y soluciones comunes de sincronización.

2. Algoritmo Secuencial

En `tp1.cpp` encontrarán un algoritmo secuencial que resuelve el problema del Árbol generador Mínimo con el nombre de `mstSecuencial`. Este algoritmo les deberá servir de base para el algoritmo paralelo que deberán desarrollar y también para verificar las respuestas de su implementación en los casos de prueba.

El pseudocódigo del algoritmo es el siguiente:

1. Crear un arbol a.
2. Inicializar todos los vertices del grafo G pintados de blanco.
3. Seleccionar un vertice v al azar de G.
4. Repetir hasta que todos los vertices de G sean negros:
   - 4.1 Pintar a v de negro.
   - 4.2 Agregar v al árbol a desde el vértice más cercano en a.[2]
   - 4.3 Pintar de gris a todos los vértices vecinos de v que sean blancos.
   - 4.4 Seleccionar como nuevo v al vértice gris más cercano al nuevo árbol a.
   - 4.5 Volver al punto 4.1.

[1] Borůvka, O. (1926). *O jistém problému minimálním.*

[2] El vértice más cercano de a es aquel que tiene un eje de menor peso que lo une a v.

3. Ejercicios

Ejercicio 1

Crear una función `void mstParalelo(Grafo *g, int cantThreads)` que resuelva el problema del Árbol Generador Mínimo utilizando `cantThreads` threads y cumpla las siguientes condiciones:

1. La estructura del grafo G y sus colores debe ser compartida.
2. Cada thread debe utilizar y hacer crecer su propio árbol a.
3. Cada thread debe pintar los vértices del grafo con un id único (en vez del color negro).
4. Debe garantizar que sólo haya contención del grafo en caso de que dos threads quieran colorear el mismo *vértice*.
5. Incrementar el número total de ejes (`incrementarTotalEjes`) debe ser una operación *wait-free*.
6. Cuando haya una colisión entre dos threads que quieran colorear el mismo vértice, deben fusionar sus árboles llamando a la función `fusionarArboles`.

La función `fusionarArboles` debe realizar lo siguiente:

1. Preservar el thread de menor id (y su correspondiente árbol) y terminar la ejecución del otro.
2. El árbol resultante debe tener los vértices y ejes de ámbos árboles.
3. Se deben colorear con el color del thread que sobreviva (el de menor id) los vértices en el grafo compartido que correspondían al otro thread.
4. La fusión entre dos threads tendrá más prioridad que buscar nuevos vértices para sumar al árbol.
5. **Sugerencias:**
   a) Una vez que un thread identifica que quiere fusionarse debe esperar que el otro thread también sea notificado de la acción antes de continuar con el algoritmo.
   b) El otro thread deberá notificarse de que tiene una fusión antes de seleccionar un nuevo vértice para pintar (puntos 3 y 4.4 del algoritmo secuencial). Este chequeo lo realizaremos usando spinlocks (¿por qué?).
   c) Noten que más de un thread podría querer fusionarse con un mismo thread, se recomienda almacenar una cola de pedidos de fusión por thread.
   d) Una vez fusionados, el thread de mayor id terminará su ejecución. Si tuviera pedidos de fusión de otros threads deberá también transferirlos al thread que sobreviva.

Ejercicio 2

Evaluar para distintos tamaños de grafos y distintas cantidades de threads el rendimiento de `mstParalelo` y `mstSecuencial`.

Ejercicio 3

Incluir casos de pruebas en la carpeta `test` que puedan correrse ejecutando diretamente instrucciones como: `make test1-secuencial`, `make test1-paralelo`, `make test-secuencial`, `make test2-paralelo` desde la carpeta principal.

4. Condiciones de entrega

Instrucciones para realizar la entrega

La entrega se realizará a través del **campus virtual**. Es muy importante tener en cuenta los siguientes requisitos para poder realizar la entrega:

- Todos los integrantes del grupo deberán estar **matriculados** en el curso de Sistemas Operativos correspondiente a este cuatrimestre. Esto quiere decir que el curso debería aparecer listado bajo "Mis cursos" en la barra de navegación de la izquierda. En caso de que algún integrante del grupo no esté matriculado o si tienen dudas al respecto, les solicitamos comunicarse *con tiempo* con los docentes.
- Deberán **informar** a los docentes **la conformación del grupo** para que pueda ser cargada en el sistema. En caso de no haberla informado presencialmente a alguno de los docentes, les solicitamos que lo hagan vía e-mail a `so-doc@dc.uba.ar` *por lo menos un día antes* de la fecha límite de entrega. El sistema **no permitirá** envíos de estudiantes que no hayan informado la conformación de sus grupos.

Para realizar la entrega, deberán acceder a la tarea "TP1 - pthreads" que se encuentra dentro de la sección "TPs" de la página del campus, y subir allí un archivo comprimido con los contenidos que se especifican más abajo.

Es suficiente que uno de los integrantes del grupo realice la entrega. Si es necesario, podrán realizar múltiples envíos, en cuyo caso el corrector solo tendrá en cuenta el último envío que se haya realizado antes de la fecha límite de entrega. Tengan en cuenta que el sistema **no aceptará** envíos pasada la fecha límite de entrega, sin excepción alguna.

Contenidos del entregable

El archivo comprimido que entreguen deberá contener **únicamente**:

- El documento del informe (en **PDF**).
- El código fuente debidamente documentado **completo y con Makefiles** de los distintos algoritmos (**NO** incluir código compilado).
- Las modificaciones al *Makefile* para correr los test agregados.

La implementación que realicen deberá estar libre de condiciones de carrera y presentar la funcionalidad descripta anteriormente.

El informe deberá ser de caracter **breve**, pero deberá incluir:

- Una explicación o justificación de qué porción del código (y cómo) resuelve cada una de las partes del algoritmo.
- Los gráficos del ejercicio 2 que deberán contar con referencias claras en los ejes y la explicación de los resultados observados.
- La explicación de los tests desarrollados para evaluar los algoritmos.