Universidad Católica
San Pablo

CIENCIA DE LA COMPUTACIÓN

Computación Paralela y Distribuida

Aplicación de Pthreads

Rodrigo Alonso Torres Sotomayor

CCOMP 8-1

Ël alumno declara haber realizado el presente trabajo de acuerdo a las normas de la Universidad Católica San Pablo"

Aplicación de Pthreads

Introducción

En este trabajo se analizará la aplicación y efectos de utilizar phtread para al paralelización en las funciones básicas de una lista enlazada. De igual manera se hará una observación sobre el funcionamiento de las barreras para pthreads, haciendo una comparación y descripción de cada una de ellas.

1. Lista enlazada

El problema de controlar el acceso a una gran estructura de datos compartida, que puede ser simplemente buscada o actualizada por los threads. Para esto se utilizará una lista enlazada, como la estructura de datos compartida, ordenada de enteros, y las operaciones de interés son Member (Pertenencia), Insertar (Insert) y Eliminar (Delete).

1.1. Requisitos de la PC

Para las pruebas de las barreras en este trabajo se utilizó una máquina virtual y las características de dicha máquina son:

- Sistema Operativo: Linux Ubuntu 20.04.
- Memoria base (RAM): 4 GB
- Procesador: 4CPUs
- Memoria VRAM: 17mb

1.2. Tiempos y Análisis

Para las pruebas se utilizó como operaciones de la lista enlazada: 32 operaciones, 80 % de búsqueda, 30 % insert y 30 % delete. Los resultados obtenidos se pueden observar en la Tabla **??**, donde la variable que varia son la cantidad de keys.

Keys	1 Thread	2 Threads	4 Threads	8 Threads
100	2.323e-03	2.389e-03	1.359e-03	2.248e-03
1000	1.23e-03	8.43e-04	8.13e-04	2.348e-03
10000	6.494e-03	6.627e-03	8.00e-03	1.332e-02

Cuadro 1: Tiempos, lista enlazada

Algoritmo	1 Thread	2 Threads	4 Threads	8 Threads
rwl	1.397321e-01	8.527613e-02	6.970286e-02	6.793094e-02

Cuadro 2: Tiempos, lista enlazada. 99.9 % Member, 0.05 % Insert, 0.05 % Delete

Para evaluar las distintas operaciones de una lista enlazada, volvemos a tomar los tiempos pero esta vez con solo un 80 % de operaciones Member, 10 % de Insert y otro 10 % de Delete.

Algoritmo	1 Thread	2 Threads	4 Threads	8 Threads
rwl	2.092807e+00	2.056543e+00	2.208202e+00	2.782938e+00

Cuadro 3: Tiempos, lista enlazada. 80 % Member, 10 % Insert, 10 % Delete

Después de analizar estos resultados se observa que al aplicar más keys y más procesos se obtienen buenos resultados, pero siempre es importante recalcar que hayq ue considerar la estructura de la máquina porque al utilizar los 8 threads a más puede generar fallas haciendo que este proceso que al inicio es óptimo perjudique su rendimiento.

2. Barreras

Las barreras son fundamentales en la programación multi-hilo, permitiendo que todos los hilos inicien una sección de código al mismo tiempo, siendo esencial para medir tiempos y lograr coherencia en la ejecución de threads. En aplicaciones donde se requiere medir el rendimiento de un programa multi-hilo, es esencial que todos los hilos comiencen y finalicen una tarea en conjunto, facilitando la obtención del tiempo tomado por el hilo más lento para completar la tarea.

2.1. Requisitos de la PC

Para las pruebas de las barreras en este trabajo se utilizó una máquina virtual y las caracteristicas de dicha máquina son:

- Sistema Operativo: Linux Ubuntu 20.04.
- Memoria base (RAM): 4 GB
- Procesador: 4CPUs
- Memoria VRAM: 17mb

2.2. Semáfora

Las semáforas son herramientas de sincronización ampliamente utilizadas en programación concurrente y multi-hilo para controlar el acceso a recursos compartidos. En el contexto de implementar barreras, las semáforas pueden ser una de las formas de lograr la sincronización necesaria para que los hilos se encuentren en un punto de espera hasta que todos hayan alcanzado dicho punto. Para implementar una barrera utilizando semáforos, se siguen pasos similares a la descripción previa de la implementación con espera activa y mutex, pero en lugar de usar un bucle de espera activa, se utilizan semáforos para coordinar el avance de los hilos.

2.3. Busy Wait con Mutex

Implica que los hilos están continuamente verificando alguna condición dentro de un bucle, sin dormir o bloquearse, y están utilizando un mutex para garantizar la exclusión mutua en el acceso a ciertas variables compartidas. En el contexto de la implementación de barreras, la idea es que los hilos están .[es]perando activamente.[en] un bucle hasta que todos los hilos han alcanzado cierto punto en el código (la barrera). Durante esta espera activa, cada hilo comprueba un contador compartido (protegido por un mutex) que indica cuántos hilos han llegado a la barrera. El uso del mutex garantiza que solo un hilo a la vez pueda actualizar el contador compartido, evitando condiciones de carrera y garantizando la consistencia de los datos compartidos.

Sin embargo, este enfoque tiene el inconveniente de que los hilos están consumiendo ciclos de CPU al verificar continuamente la condición en el bucle de espera activa, lo que puede ser ineficiente en términos de utilización de recursos. Además, si hay más hilos que núcleos de CPU, esto puede resultar en una competencia excesiva por los recursos de la CPU y afectar el rendimiento del sistema.

2.4. Variables de Condición

Las variables de condición son herramientas esenciales en programación concurrente y multi-hilo que permiten a los hilos esperar hasta que se cumpla una cierta condición. En el contexto de implementar barreras, las variables de condición pueden ser utilizadas para coordinar la sincronización de los hilos en un punto específico del programa antes de permitirles continuar. Este enfoque utilizando variables de condición reduce la necesidad de espera activa, mejorando la eficiencia y reduciendo el consumo de recursos de la CPU al permitir que los hilos estén en un estado de espera más eficiente hasta que se cumpla la condición deseada.

2.5. Resultados y Análisis

Para las pruebas de la barreras se hizo la comparación de los tiempos de ejecución mientras iba incrementando el uso de threads. Los resultados obtenidos se pueden visualizar en la Tabla 4.

Haciendo una comparación más precisa entre la Semáfora y las Variables de Condición ya que sus resultados se asemejan más se observa como la semáfora es más constante a medida que aumentan los threads.

	1 Thread	2 Threads	3 Threads	4 Threads	5 Threads
Semáfora	9.0885e-04	2.5479e-02	3.05531e-02	6.0910e-02	6.3595e-02
Busy Wait	9.2292e-04	2.0430e-03	1.9321e-03	6.99838	6.58866
Variables de Condición	9.07898e-04	5.0458e-02	5.5891e-02	5.7955e-02	8.0966e-02

Cuadro 4: Tabla de tiempos en segundos, Barreras

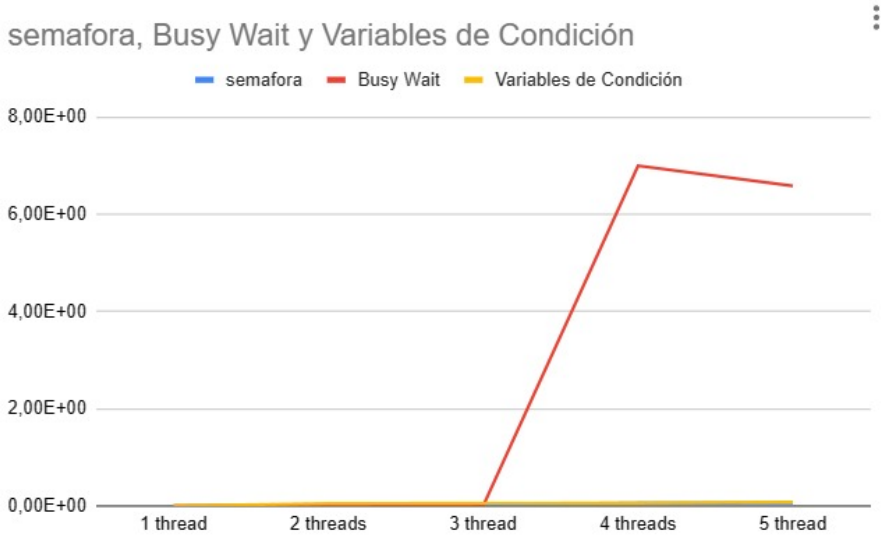

Figura 1: Gráfico de lineas, Semáfora, Busy Wait y Variables de Condicion

Se concluye que utilizando la barrera de semáfora se pueden obtener los mejores resultados respecto a tiempo y el busy wait mientras más threads más tiempo de ejecución le tomaba. Pero es importante recalcar que los tiempos de la aplicación de estas barreras puede variar según la situación. Ya que cada semáfora está diseñada para situaciones específicas.

3. Código de la implementación

Los códigos utilizados para estas pruebas estan en: https://github.com/RodATS/ComputacionParalela.git

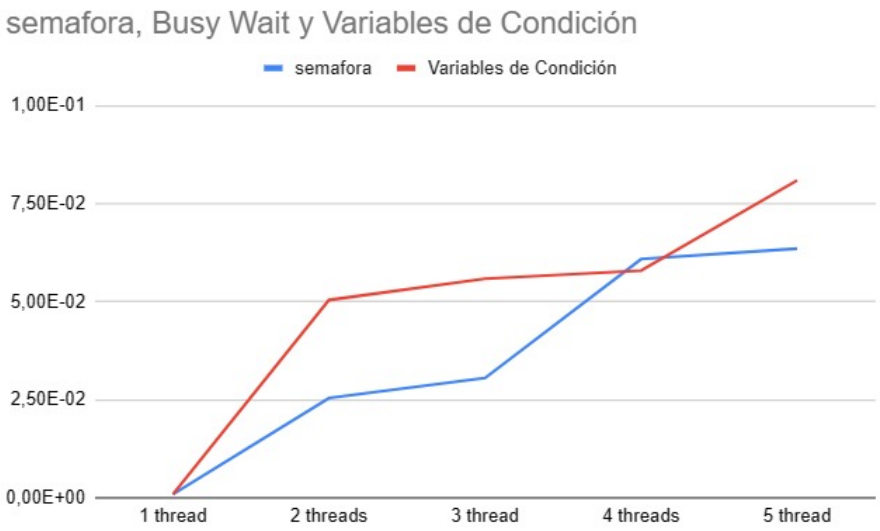

Figura 2: Gráfico de lineas, Semáfora y Variables de Condicion