Control II: Ejecución Especulativa - Lenguajes de Programación II

Andrés González, andres.gonzalezvi@estudiantes.uv.cl

Lorena Uribe, lorena.uribe@estudiantes.uv.cl

Repositorio: https://github.com/Uribe22/Lenguaje-de-Programaacion-.git

Profesor: Alonso Inostrosa Psijas

Se implementó en Go un patrón de ejecución especulativa que compara una versión concurrente utilizando goroutines con channels contra una versión secuencial. El objetivo fue evaluar el rendimiento de ambas versiones utilizando un conjunto de cálculos costosos, que se ejecutan en paralelo en la versión especulativa. Para medir las mejoras, se ejecutó un benchmark con 30 repeticiones para cada estrategia. Los resultados mostraron que la ejecución especulativa no mejora significativamente los tiempos de ejecución para las configuraciones probadas.

Tabla 1 — Parámetros de prueba

Prueba	**blockData**	**dificultad**	**maxPrimos**	**n**	**umbral**
1	blockchain_data	4	5000	15	200
2	bloque_genesis	5	5000	15	200
3	datos_prueba	3	100000	15	5000

Tabla 2 — Resultados (promedios)

Prueba	**Especulativo (ms)**	**Secuencial (ms)**	**Speedup**
1	17	17	1.00
2	1410	1431	1.01
3	4	3	0.75

Este consiste en un programa que implementa un patrón de ejecución especulativa en Go, un lenguaje con soporte nativo para la concurrencia mediante goroutines y channels. El objetivo fue comparar el rendimiento de las versiones concurrente y secuencial de tres cálculos costosos:

- Simulación de Proof of Work: Simula el proceso de minería de bloques en una blockchain.
- Búsqueda de Números Primos: Encuentra todos los números primos hasta un número máximo.
- Cálculo de la Traza de una Matriz: Calcula la traza de una multiplicación de matrices de dimensión n.

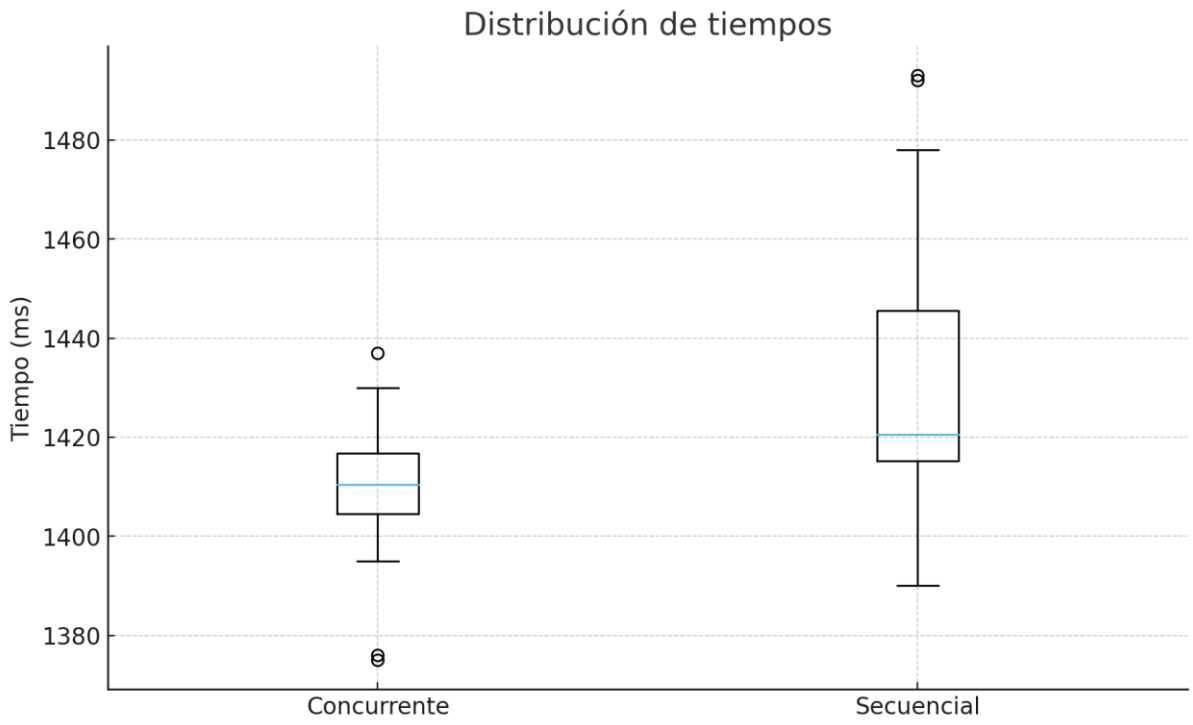
Funcionamiento del Programa

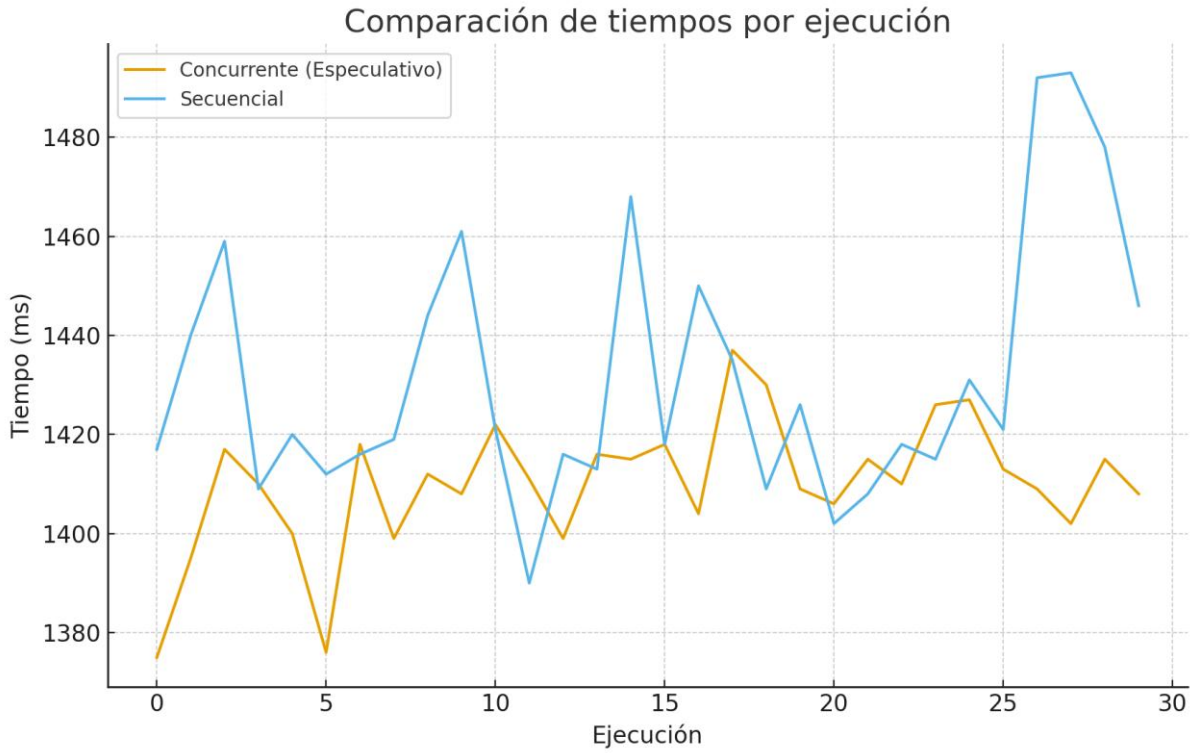
- Concurrente: Las funciones de Proof of Work y Búsqueda de Primos se ejecutan en goroutines por separado, comunicándose con el hilo principal a través de channels. El cálculo de la traza de la matriz se realiza en el hilo principal y es usado para decidir cuál de las dos ramas concurrentes (Proof of Work o Primos) es la ganadora. La rama perdedora es cancelada antes de completar su ejecución.
- Secuencial: El cálculo de la traza de la matriz se realiza primero y luego se ejecuta el cálculo de la rama seleccionada de manera secuencial.

Modo Benchmark

Se implementó un modo benchmark que ejecuta ambos enfoques (concurrente y secuencial) 30 veces, para obtener el tiempo promedio de cada ejecución. Esto permite comparar la efectividad de ambos enfoques en términos de tiempo.

Análisis de Resultados





Observaciones

- Diferencia de Tiempos: En las pruebas realizadas, la diferencia en los tiempos de ejecución entre el enfoque concurrente y el secuencial fue pequeña. El speedup observado en la mayoría de las pruebas fue cercano a 1, lo que indica que el uso de goroutines no proporcionó mejoras significativas.
- Ejecución Especulativa: Aunque en algunos casos el tiempo fue ligeramente mejor (como en el Test 2), en otros casos el enfoque concurrente resultó en un overhead que hizo que el tiempo de ejecución fuera incluso mayor que en el enfoque secuencial.
- Tareas Rápidas: Para tareas que se ejecutan rápidamente (como en Test 3), la creación de goroutines y la sincronización de channels añadieron un overhead significativo, lo que resultó en un speedup menor que 1.

Conclusiones

No hubo una mejora significativa en el rendimiento de los cálculos utilizando ejecución especulativa, especialmente cuando la condición de selección es costosa.

La ejecución secuencial sigue siendo una opción competitiva, ya que no incurre en el overhead de sincronización y control de goroutines.

La concurrencia no siempre resulta en mejoras de rendimiento para este tipo de tareas, especialmente cuando la tarea principal de decisión no es intensiva en tiempo y el costo de gestión de hilos se convierte en un factor limitante.