Implementación de Forks y Threads: análisis y discusión.

Este informe tiene como objetivo realizar un análisis comparativo del rendimiento de la multiplicación de matrices utilizando distintas estrategias: Forks y Pipes en C++, y Threads en Java. A lo largo del documento, se abordarán y responderán preguntas relacionadas con el análisis de estas implementaciones, con el fin de evaluar su eficiencia y comportamiento en distintos escenarios.

Descripción del equipo: David Kripper (Forks y pipes)

Se utilizó un MacBook Air M2 (2022) con las especificaciones siguientes:

- **Procesador:** Chip M2 Apple con 4 núcleos de rendimiento @3.49 GHz y 4 núcleos de eficiencia @2.42 GHz. GPU de 8 núcleos. Neural Engine de 16 núcleos. 8 hilos de ejecución por núcleo (núcleos de rendimiento y de eficiencia).
- **Memoria RAM:** 8 GB de memoria unificada LPDDR5.
- **Sistema Operativo:** macOS Sequoia 15.4.1.
- **Arquitectura del sistema:** 64 bits.

Descripción del equipo: Tomás González (Threads)

Se utilizó un HP Pavilion x360 Convertible 14-dy0xxx con las siguientes especificaciones:

- **Procesador:** 11th Gen Intel(R) Core(TM) i3-1125G4 @2.0GHz, 1997 Mhz, 4 procesadores físicos, 8 procesadores lógicos. GPU integrada Intel(R) UHD Graphics.
- **Memoria:** 8GB de memoria DDR4.
- **Sistema operativo:** Microsoft Windows 11 Home Single Language.
- **Arquitectura del sistema:** 64 bits.

Preguntas de análisis:

1. ¿Cuál de las dos implementaciones tuvo un mejor rendimiento en términos de tiempo de ejecución? ¿A qué crees que se debe esto?

   R: La implementación que mostró un mejor rendimiento en términos de tiempo de ejecución corresponde a la de Forks y Pipes. Ésta siempre obtuvo tiempos de ejecución iguales o menores a la implementación con threads. Lo anterior se puede deber a los siguientes puntos: (1) Crear procesos con fork en sistemas modernos tiene un costo muy bajo gracias al mecanismo de copy-on-write, por lo que la creación apenas impacta en problemas con pocas multiplicaciones. (2) Los threads, al compartir memoria, necesitan una coordinación entre sí, lo que añade latencia adicional al realizar operaciones de multiplicación.

2. ¿Se observó alguna diferencia significativa en el uso de recursos del sistema (CPU, RAM) entre ambas soluciones?

   R: No hubo diferencias significativas en el uso de recursos entre ambas soluciones. Para la implementación de Forks + Pipes hubo un consumo un poco mayor en CPU y RAM pero no considerable en comparación a la implementación con threads, la cual tuvo un rendimiento estable y mostró un consumo de CPU liviano.

3. ¿En qué escenarios considera más adecuado el uso de procesos (forks) frente a hilos (threads)?

   R: En escenarios de matrices pequeñas o de tamaño intermedio (hasta 6×6), la implementación con forks y pipes demostró ser tan rápida o incluso más rápida que la basada en threads. Por ello, cuando las dimensiones no superan este rango, conviene emplear procesos, pues en todos los casos registrados el tiempo de ejecución fue igual o inferior al de la versión con hilos.

4. ¿Qué estrategias de optimización aplicarías al programa con menor rendimiento para que iguale o supere la eficiencia del programa mejor evaluado?

   R: Para mejorar y/o igualar el rendimiento de la implementación basada en Forks + Pipes, la implementación basada en threads puede realizar las siguiente optimizaciones: (1) Reutilizar hilos en vez de crear uno por fila: el coste de crear y luego hacer join() es notable cuando las matrices tienen dimensiones pequeñas. En lugar de lo anterior se puede crear un grupo de hilos una única vez, y que estos se mantengan vivos durante toda la ejecución del programa. (2) Compilar con optimizaciones nativas de Java, como lo son las librerías nativas JAMA o ND4J, las cuales implementan la multiplicación de matrices en C++. Otra optimización es ejecutar con flags adecuados de HotSpot (p. ej., activar escape analysis, tiered compilation). (3) Transponer la matriz B para mejorar localidad de datos: se puede transponer la matriz B justo después de leerla ya que de esa manera, ambas lecturas de matrices se hacen por filas contiguas en memoria, reduciendo fallos de caché y acelerando el bucle interno.