Práctica 7

Introducción al Análisis de Rendimiento en OpenMP

Jordi Blasco Lozano
Computación de alto rendimiento
Grado en Inteligencia Artificial

Indice:

1. Introducción

La practica siguiente se desarrolla a partir de unos códigos proporcionados. El objetivo principal de la práctica es comprender y reflexionar sobre las métricas rendimiento obtenidas al analizar los tiempos de ejecución mediante el uso de las distintas técnicas de paralelización de OpenMP. Como el código para realizar las tablas que se pedían era muy mecánico se me ocurrió la idea de generar un único código con 5 funciones principales, cada ejercicio una función. Para los ejercicios del 2 al 5 usé 4 voids que imprimiesen las tablas requeridas. Mientras que para el ejercicio 1 use una función que además de imprimir la tabla devolviera un vector con los resultados para poder compáralos con el ejercicio 4.

2. Desarrollo

Para realizar con orden y criterio todos los ejercicios seguiremos siempre la misma práctica.

introduccion resultados conclusiones

Ejercicio 1:

En este primer ejercicio se realiza un análisis de rendimiento aplicado a un problema de suma acumulada de una única variable dentro de un bucle paralelizado. La práctica se centra en medir el tiempo de ejecución del programa utilizando distintos números de hilos (1, 2, 4, 8 y 16) para evaluar el impacto de la paralelización. A partir de las mediciones de tiempo se calcula el **speed-up** y la **eficiencia**.

- Tiempo de ejecución **(T_n)**: Se mide el tiempo que tarda el programa en ejecutarse con **N** hilos usando omp_get_wtime().
- Speed-up **(S)**: Se define como la relación entre el tiempo de ejecución secuencial (1 hilo) y el tiempo con **N** hilos, es decir: $\mathbf{S} = \frac{T_1}{T_N}$
- Eficiencia **(E)**: Mide el grado de aprovechamiento de los hilos, calculado como $\boldsymbol{E} = \frac{S}{N}$

El objetivo es observar cómo varían estas métricas al aumentar el número de hilos y evaluar si el programa escala de forma lineal o si existen efectos de sobrecarga que afecten la eficiencia.

Hilos (N)	T_N (s)	Speed-up	Eficiencia
1	25.0426	1.00	1.00
2	12.6149	1.99	0.99
4	6.4653	3.87	0.97
8	3.3495	7.48	0.93
16	1.8108	13.83	0.86

Los resultados obtenidos muestran una mejora significativa en el tiempo de ejecución al aumentar el número de hilos. Con 1 hilo, el tiempo de ejecución fue de aproximadamente 25.04 segundos, reduciéndose progresivamente a 1.81 segundos con 16 hilos. El speed-up casi se comporta de forma lineal (por ejemplo, se alcanza un speed-up de 7.48 con 8 hilos y 13.83 con 16 hilos), lo que indica que la porción paralelizable del código es sustancial. Sin embargo, se observa una ligera disminución en la eficiencia a medida que se incrementa el número de hilos (pasando de un 100% con 1 hilo a un 86% con 16 hilos), reflejando el costo del overhead asociado a la gestión de la paralelización.

En resumen, el ejercicio demuestra que, a pesar de las inevitables pérdidas de eficiencia debidas a la sobrecarga de paralelización, el programa escala de manera favorable cuando se aumenta el número de hilos, haciendo un uso eficaz de los recursos disponibles para acelerar el cómputo.

Ejercicio 2:

En este ejercicio se evalúa el impacto de tres técnicas de sincronización para actualizar una variable compartida en un bucle: **critical**, **atomic** y **reduction**. La idea es analizar cómo varía el tiempo de ejecución al incrementar el número de hilos (1, 2, 4, 8 y 16) en cada caso.

- En la técnica **critical**, se utiliza una sección crítica para proteger la actualización de la variable compartida. Esto implica que, a medida que se añaden más hilos, la contención aumenta notablemente, ya que cada hilo debe esperar su turno para acceder a la sección, generando cuellos de botella.
- Con la técnica **atomic**, se emplean operaciones atómicas para actualizar la variable. Aunque son más ligeras que las secciones críticas, también sufren de contención al incrementarse los hilos, lo que se refleja en tiempos de ejecución crecientes.
- Por último, la técnica **reduction** permite que cada hilo acumule su resultado de forma local y, posteriormente, se combinen estos resultados. Este enfoque minimiza la contención y reduce significativamente el overhead asociado con la sincronización.

Hilos	Critical(s)	Atomic(s)	Reduction(s)
1	4.2256	3.9355	2.5182
2	33.6221	21.6872	1.2802
4	84.2287	41.7595	0.6652
8	143.8371	59.0352	0.3556
16	221.1664	118.6365	0.1993

Los resultados obtenidos reflejan de forma clara el impacto de la contención en las técnicas critical y atomic a medida que se aumenta el número de hilos.

Podemos observar que, al aumentar la cantidad de hilos, el tiempo de ejecución con critical se dispara de 4s a 221s, lo cual se debe a la alta contención que se genera cuando múltiples hilos intentan acceder a la misma sección crítica de forma secuencial. De manera similar, la técnica atomic también presenta incrementos significativos en el tiempo, aunque su overhead es menor que en el caso de critical.

En contraste, reduction muestra una mejora dramática en el rendimiento. Por ejemplo, al utilizar 16 hilos, la técnica reduction tarda aproximadamente 0.1993 s, lo que representa una mejora de alrededor de **x1100** en comparación con critical y de cerca de **x600** respecto a atomic. Esta notable diferencia se debe a que reduction evita la contención al permitir que cada hilo realice sus operaciones de forma local, combinando los resultados de manera eficiente al final de la ejecución.

En conclusión, mientras que las técnicas **critical** y **atomic** sufren de un aumento de tiempo de ejecución significativo debido a la contención en entornos con muchos hilos, **reduction** demuestra ser la estrategia más adecuada para minimizar el overhead y aprovechar al máximo la paralelización en este tipo de operaciones.

Ejercicio 3:

En este ejercicio se analiza el impacto de distintas estrategias de distribución de la carga (**schedules**) sobre el rendimiento de un algoritmo que ejecuta bloques con cargas desiguales. Se comparan tres políticas de schedule: **static**, **dynamic** y **guided**, evaluadas con 4 y 8 hilos. La idea es estudiar cómo la elección del schedule afecta la distribución de la carga entre hilos y, por ende, el tiempo total de ejecución. Particularmente, se observa que al aumentar el número de hilos, la política Dynamic mejora notablemente su rendimiento debido a la mayor flexibilidad para reasignar tareas en caso de desbalance, mientras que static se ve limitada, ya que asigna de forma fija los bloques a cada hilo sin considerar las diferencias en la carga de trabajo. Los primeros resultados no me convencían para poder explicar de forma coherente cómo funciona cada técnica, por lo que aumente un 0 la carga por cada bucle de complejidad variable.

Schedule	4 hilos (s)	8 hilos (s)
static	3.2813	3.5169
dynamic	4.2263	2.3921
guided	3.4217	3.2832

Los resultados muestran diferencias significativas entre las políticas de schedule al aumentar el número de hilos en un escenario con cargas desiguales:

- **Static**: Con 4 hilos se obtiene un tiempo de 3.28s, pero al aumentar a 8 hilos el tiempo se incrementa ligeramente a 3.51s. Esto se debe a que la asignación fija de bloques en static no se adapta a las variaciones en la carga, generando un estancamiento del rendimiento cuando algunos hilos reciben más trabajo que otros.
- **Dynamic**: Con 4 hilos se obtiene un tiempo de 4.22s, mayor que con 8 hilos debido al **overhead** de sincronización al repartir las tareas de forma flexible mientras el código se ejecuta. Al aumentar a 8 hilos, este overhead se reduce, permitiendo que los hilos libres asuman más trabajo aprovechando todos los recursos al aumentar la complejidad del problema mejorando el tiempo de ejecución hasta los 2.39s.
- **Guided**: La política guided se ubica en un punto intermedio, con tiempos de 3.42s y 3.28s para 4 y 8 hilos respectivamente, reflejando una mejora moderada al aumentar los hilos, pero sin alcanzar la eficiencia observada con dynamic.

En resumen, en contextos de cargas desiguales, **dynamic** demuestra ser la opción más eficaz cuando se utilizan 8 hilos, ya que se adapta mejor a la variabilidad de la carga, mientras que **static** se ve limitada por su asignación fija y no logra beneficiarse del aumento en el número de hilos.

Ejercicio 4:

En este ejercicio se contrasta el **speed-up real** obtenido de la ejecución paralela con **el speed-up teórico** calculado mediante la **Ley de Amdahl**. Se asume que el **85%** del código es paralelizable **(p=0.85)**, lo que permite estimar el límite teórico del rendimiento y compararlo con las mediciones reales (tomadas del Ejercicio 1) para configuraciones con 4 y 8 hilos.

p	**0.85**
Speed-up teórico (hilos infinitos)	6.66667
Speed-up teórico (4 hilos)	2.75862
Speed-up teórico (8 hilos)	3.90244

p	**real**
Speed-up empirico (4 hilos)	3.86779
Speed-up empirico (8 hilos)	7.4835

La Ley de Amdahl asume que una fracción **p** del código es paralelizable, y en este caso se estimó **p = 0.85**. Bajo esta suposición, se espera que el speed-up máximo (con hilos infinitos) sea de 6.66667, y que con 4 y 8 hilos se obtengan speed-ups teóricos de 2.75862 y 3.90244, respectivamente. Sin embargo, los resultados empíricos muestran que con 8 hilos el speed-up alcanza 7.4835, superando incluso el valor teórico para hilos infinitos.

Este comportamiento puede explicarse por varios factores:

- **Subestimación del grado de paralelismo**: Es posible que la fracción paralelizable real $\mathbf{p_{real}}$ sea mayor que el 85% estimado. Si el código tiene menos secciones secuenciales o si estas se ejecutan más rápido en la práctica, el rendimiento puede superar el límite teórico calculado con un p inferior.
- **Mejoras en la caché y la localidad de los datos**: Con un mayor número de hilos, la distribución de datos y la eficiencia del uso de la memoria caché pueden mejorar, reduciendo el tiempo de ejecución más de lo previsto por el modelo teórico.
- **Optimización del scheduler y comportamiento del sistema:** Los mecanismos de gestión de hilos en los sistemas modernos pueden optimizar la distribución de tareas de manera que se minimicen los tiempos de espera y se aprovechen mejor los recursos, generando un rendimiento superior al estimado.

En resumen, la diferencia entre el **speed-up teórico** y el **empírico** puede deberse a que el modelo de Amdahl es una aproximación simplificada que no contempla todas las optimizaciones posibles. Además, si la fracción paralelizable real es mayor que la asumida teóricamente, se obtiene un rendimiento mayor al previsto, lo que explica cómo el speed-up empírico con 8 hilos puede superar incluso el límite teórico calculado para hilos infinitos.

Ejercicio 5:

El objetivo del Ejercicio 5 es analizar la escalabilidad débil, es decir, evaluar si el tiempo de ejecución se mantiene aproximadamente constante al aumentar de manera proporcional tanto la carga de trabajo como el número de hilos. En un escenario ideal de escalabilidad débil, el incremento en el número de elementos procesados (proporcional al número de hilos) no debería alterar significativamente el tiempo total de ejecución.

Hilos	Elementos procesados	Tiempo(s)
1	100000000	0.2556
2	200000000	0.2558
4	400000000	0.2587
8	800000000	0.2705

Los resultados demuestran que, al duplicar tanto la carga de trabajo como el número de hilos, el tiempo de ejecución se mantiene prácticamente constante. Esto indica una buena escalabilidad débil del sistema, ya que el aumento de recursos se traduce en un manejo proporcional de la carga sin incrementos significativos en el tiempo de procesamiento. Las pequeñas variaciones observadas pueden atribuirse a la sobrecarga inherente en la gestión de hilos y la sincronización.