Universidad Nacional de La Plata
Facultad de Informática
Sistemas Paralelos

Informe Trabajo Práctico 2 | Optimización de Algoritmos Secuenciales y Paralelos

Por Franco Kumichel y Norberto Ariel Chaar

Introducción

Se propone realizar un informe que describa brevemente las soluciones planteadas, análisis de resultados y conclusiones. Incluyendo el detalle del trabajo experimental (características del hardware y del software usados, pruebas realizadas, etc), con los tiempos de ejecución, SpeedUp y Eficiencia en tablas. El análisis de rendimiento se realiza tanto en forma individual a cada solución paralela como en forma comparativa.

Características del hardware y software usados:

Clúster provisto por la Cátedra de Sistemas Paralelos de la Facultad de Informática UNLP, el cual posee las siguientes características:

- 8GB de RAM
- Dos procesadores Intel XeonE5405

El procesador Intel XeonE5405 posee las siguientes características:

- Frecuencia básica: 2,0GHz
- Caché: 12 MB L2 Cache
- Número de cores físicos: 4

- Hilos por core: 4
- Velocidad del bus: 1333 MHz

Enunciado

Dada la siguiente expresión:

$$R = \frac{(MaxA \times MaxB - MinA \times MinB)}{PromA \times PromB} \times [A \times B] + [C \times D]$$

- Donde A, B, C, D y R son matrices cuadradas de NxN con elementos de tipo double.
- MaxA, MinA y PromA son los valores máximo, mínimo y promedio de la matriz A, respectivamente.
- MaxB, MinB y PromB son los valores máximo, mínimo y promedio de la matriz B, respectivamente.

Desarrolle 3 algoritmos que computen la expresión dada:

1. Algoritmo secuencial optimizado
2. Algoritmo paralelo empleando Pthreads
3. Algoritmo paralelo empleando OpenMP

Mida el tiempo de ejecución de los algoritmos en el clúster remoto. Las pruebas deben considerar la variación del tamaño de problema (N={512, 1024, 2048, 4096}) y, en el caso de los algoritmos paralelos, también la cantidad de hilos (T={2,4,8}). Por último, recuerde aplicar las técnicas de programación y optimización vistas en clase.

La solución implementada cuenta con los 3 algoritmos solicitados, siendo los siguientes:

- **ecuacion_secuencial.c:** Utiliza la misma estructura implementada en la anterior entrega, aunque fue adaptada para complementarse con la solución dada para esta entrega, además de corregir aspectos apuntados en la primera entrega por parte de quien corrigió el primer informe.
- **ecuacion_pthreads.c:** algoritmo paralelo implementado utilizando Pthreads
- **ecuacion_openMP.c:** algoritmo paralelo implementado utilizando OpenMP

Algoritmo Secuencial

En base al algoritmo implementado en la entrega anterior, se han realizado las siguientes modificaciones en base a las correcciones hechas por la cátedra y además modificaciones realizadas para que el algoritmo se adapte a las soluciones implementadas para el problema actual.

- El cálculo del mínimo, máximo y promedio de las matrices A y B se realizan en diferentes ciclos for en lugar de calcular todo en uno mismo para así aprovechar el principio de localidad de datos.
- Para la multiplicación de matrices, se decidió almacenar en variables auxiliares valores pre calculados de los índices para no tener que hacer en cada iteración la multiplicación.
- Se optó por realizar la multiplicación de matrices con funciones. Aunque sin el uso de funciones el tiempo de ejecución es menor, consideramos que es mínimo comparado con tener un algoritmo más modular y legible.

Algoritmo con Pthreads

Hay algunos aspectos relevantes que se tuvieron en cuenta a la hora de realizar el algoritmo:

- Se trató de utilizar la menor cantidad de barreras posible. Esto es debido a que un uso indiscriminado de las mismas puede aumentar considerablemente el tiempo de ejecución
- Para la búsqueda de máximos, mínimos y suma de matrices se utilizó un solo lock en lugar de 3. Esto considerando que dentro del lock no se realiza mucho cómputo y el hecho de utilizar 3 si bien explotaría más el paralelismo, afectaría al tiempo de ejecución.
- Dado que el valor de RP lo van a necesitar todos los hilos para realizar la multiplicación de todos los elementos de AB con RP y que este no demanda mucho cómputo, se decidió que cada hilo calcule dichos valores.

Algoritmo con OpenMP

A la hora de la implementación del algoritmo con OpenMP, se consideraron los siguientes aspectos:

- **Utilización de directiva for:** para que la distribución de datos entre hilos no sea responsabilidad del programador.
- **Utilización de cláusula reduction:** se utilizó en este caso para el cálculo de máximo, mínimos y sumas con el fin de no tener que utilizar locks o semáforos.
- **Utilización de directiva single:** Se incluye para el cálculo de RP y promedios para que un solo hilo realice el cálculo de dichos valores, ya que al agregar esta directiva nos aseguramos que un solo hilo (el primero que llegue) ejecutará el código contenido dentro.
- **Utilización de directivas nowait:** Se incluye para que no sea necesario esperar a que otros hilos finalicen sus respectivas iteraciones para continuar con la siguiente. Hay que

tener en cuenta que OpenMP posee barreras implícitas en regiones paralelas, y esta directiva justamente las evita.

Resultados

Tiempos de ejecución

Para la ejecución de los 3 algoritmos se utilizó la optimización O3 y tamaño de bloque 128 ya que como se vio en la entrega anterior con estas características fueron con las que se vieron los resultados más favorables con respecto al tiempo de ejecución.

Tiempos de ejecución para el algoritmo secuencial:

N	512	1024	2048	4096
Tiempo ejecución	0,450629	3,559687	28,362179	226,584067

Tiempos de ejecución para el algoritmo paralelo con Pthreads:

Threads/N	512	1024	2048	4096
2	0,235477	1,868825	14,753258	116,868470
4	0,120781	0,947426	7,436789	59,088448
8	0,066971	0,487155	3,799490	30,618113

Tiempos de ejecución para el algoritmo paralelo con OpenMP:

Threads/N	512	1024	2048	4096
2	0,226534	1,780832	14,067375	113,04119
4	0,115783	0,894797	7,083801	56,858088
8	0,063833	0,463003	3,639164	29,658581

Cálculo de Speedup y Eficiencia

El **speedup** es la relación entre el tiempo de ejecución de una tarea en un sistema secuencial y el tiempo de ejecución de la misma tarea en un sistema paralelo. Para calcular el mismo se utiliza la siguiente relación:

$$S_p(n) = \frac{T_s(n)}{T_p(n)}$$

La **eficiencia** mide la relación entre el tiempo empleado y la cantidad de recursos utilizados para realizar una tarea específica en un entorno paralelo. Una alta eficiencia indica que el sistema está aprovechando de manera óptima sus recursos para completar las tareas. Este se calcula de la siguiente manera:

$$E_p(n) = \frac{S_p(n)}{S_{optimo}}$$

Hay que tener en cuenta que la arquitectura del nodo del cluster es homogénea, por lo que el Speedup óptimo será igual a la cantidad de threads que le asignemos al código paralelo.

Algoritmo paralelo con Pthreads

Speedup

P/N	512	1024	2048	4096
2	1,91	1,90	1,92	1,93
4	3,73	3,75	3,81	3,83
8	6,72	7,30	7,46	7,40

Eficiencia

P/N	512	1024	2048	4096
2	0,955	0,95	0,96	0,965
4	0,9325	0,9375	0,9525	0,9575
8	0,84	0,9125	0,9325	0,925

Algoritmo paralelo con OpenMP

Speedup

P/N	512	1024	2048	4096
2	1,98	1,99	2,01	2,00
4	3,89	3,97	4,00	3,98
8	6,72	7,30	7,79	7,63

Eficiencia

P/N	512	1024	2048	4096
2	0,99	0,995	1,005	1
4	0,9725	0,9925	1	0,995
8	0,84	0,9125	0,97375	0,95375

Análisis de los resultados obtenidos

Observando las tablas anteriores, podemos concluir que:

- En los tiempos de ejecución se puede observar que en los que se utilizó el algoritmo paralelo con OpenMP son levemente menores que cuando se utilizó el algoritmo paralelo con Pthreads. En particular se pueden observar que los menores tiempos de ejecución se dan con 8 hilos en ambos algoritmos.

- Ambos algoritmos poseen un rendimiento similar. Además podemos observar que la eficiencia de ambos algoritmos aumenta a medida que crece el tamaño de la matriz y empeora cuanto más hilos son utilizados. Además podemos observar que la eficiencia es cercana a 1 en la mayoría de los casos. Esto es debido a que el problema es altamente paralelizable.

Por lo tanto, si se desea obtener la mayor eficiencia posible para matrices de tamaños 512, 1024, 2048 y 4096, lo mejor es utilizar dos hilos y OpenMP. Si en cambio se quieren lograr el menor tiempo de ejecución para todos los tamaños de matrices dados, es recomendable utilizar 8 hilos pero sacrificando eficiencia.