Modelo de Memoria Compartida

Informe entrega 2

Sistemas Distribuidos y Paralelos | Ingeniería en Computación

01515/4 Arreche, Cristian

01528/0 Borini, Ángel

06 de mayo de 2020

Facultad de Informática

Universidad Nacional de La Plata

Nota: Todas las ejecuciones fueron realizadas en el cluster provisto por la cátedra, para comparar las distintas versiones sobre la misma arquitectura.

01. Matriz ABC - Secuencial

*Realizar la multiplicación de matrices cuadradas ABC de N*N. Debe ejecutarse el algoritmo secuencial y con 4 y 8 hilos.*

El problema planteado puede tener distintas soluciones secuenciales que pueden llegar a un mismo resultado correcto. El objetivo es poder realizar una comparación con el algoritmo paralelo, por lo que es necesario elegir el algoritmo secuencial más rápido. Por este motivo, en la versión secuencial se tuvieron en cuenta todas las mejoras realizadas en el ejercicio 1 de la práctica 1.

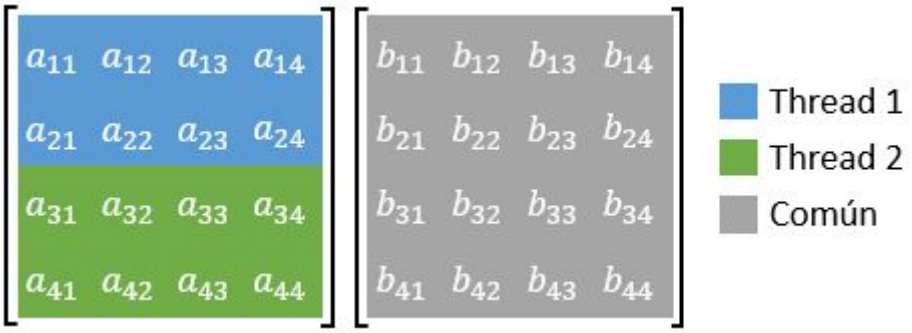
El algoritmo para la versión secuencial se implementó de manera independiente, y los tiempos obtenidos fueron los siguientes:

	TAMAÑO N		
PROCESADORES	N = 512	N = 1024	N = 2048
T = 1	1.952274	15.931320	127.489117

Tiempos de ejecución ABC_secuencial.c

01. Matriz ABC - Pthreads

Una forma de paralelizar el producto matricial es dividir las operaciones en filas y columnas entre los hilos. Si usamos matrices de dimensiones NxN, el total de operaciones de producto punto entre filas y columnas es igual a N*N. Si dividimos estas operaciones en partes iguales entre T hilos de ejecución, cada hilo llevará a cabo (N*N)/T operaciones.



Ejemplo de división de operaciones para 2 Threads

En Pthreads, se debe implementar la función de comportamiento de cada hilo, de esta manera, todos los hilos ejecutarán exactamente la misma rutina, pero los elementos a los que accederán dependen del ID de cada hilo, el cual será pasado como argumento a la rutina de ejecución.

Los tiempos de ejecución obtenidos para la versión paralela en Pthreads son los siguientes (también se incluyen nuevamente los tiempos obtenidos en la versión secuencial para apreciar la comparación):

	TAMAÑO N		
PROCESADORES	**N = 512**	**N = 1024**	**N = 2048**
T = 1	1.952274	15.931320	127.489117
T = 4	0.513714	4.187032	33.565554
T = 8	0.257757	2.092490	16.792256

Tiempos de ejecución ABC_pthreads.c

Como podemos observar, la diferencia en tiempos de ejecución entre la versión secuencial y la paralela con 4 y 8 threads respectivamente, es muy significante.

A continuación se muestra una tabla con los valores de varios speedup, el cual es un indicador de rendimiento de un algoritmo paralelo respecto a su versión secuencial; analiticamente se describe como:

Speedup = Tiempo de ejecución secuencial / Tiempo de ejecución paralelo.

El máximo valor teórico alcanzable es “P”, siendo este el número de procesadores utilizados.

	TAMAÑO N		
PROCESADORES	**N = 512**	**N = 1024**	**N = 2048**
T = 4	3.800313015	3.804919571	3.798214389
T = 8	7.574087222	7.613570435	7.592137531

Cálculos de Speedup

Por otra parte tenemos la tabla de eficiencia, otra métrica de rendimiento, cuyo valor teórico ideal y máximo es “1”. Se calcula como:

Eficiencia = Speedup / P.

	TAMAÑO N		
PROCESADORES	**N = 512**	**N = 1024**	**N = 2048**
T = 4	0.9500782538	0.9512298928	0.9495535973
T = 8	0.9467609028	0.9516963044	0.9490171914

Cálculos de eficiencia

Analizando las tablas se puede concluir en que, tanto en speedup como eficiencia, se obtuvieron los valores deseados ya que se aproximan a los ideales establecidos anteriormente.

01. Matriz ABC - OpenMP

A diferencia de Pthreads, la distribución de trabajo entre los hilos es interna en OpenMP. Por lo que no se implementa una rutina de comportamiento para los hilos, sino que se hace uso de la directiva *'parallel'* junto con otras cláusulas para crear los hilos y establecer el flujo de ejecución.

En esta instancia optamos por hacer uso de una única directiva *'parallel'* entre ambos bucles del producto matricial, ya que internamente cuando OpenMP encuentra el fin de un bloque *'parallel'* envía a los hilos a dormir, y cuando encuentra la siguiente directiva *'parallel'* despierta a los hilos inmediatamente, por lo tanto, se ocasionaría un overhead mínimo pero innecesario.

Para la distribución de trabajo entre los hilos, junto con la directiva *'parallel'*, se hizo uso de la cláusula *'omp for',* utilizada de forma estática (por defecto), donde las iteraciones del for se distribuyen proporcionalmente entre los hilos. En este caso solo se paraleliza el *'for'* más externo, sin uso del *'collapse()'.* Esto es porque el uso del *'collapse'* distribuye más las operaciones de la iteración, pero al realizar las pruebas con ambas versiones, notamos que con esa nueva distribución se pueden estar causando más fallos de caché y los tiempos de ejecución aumentan.

Los tiempos de ejecución obtenidos para la versión paralela en OpenMP son los siguientes (también se incluyen nuevamente los tiempos obtenidos en la versión secuencial para apreciar la comparación):

	TAMAÑO N		
PROCESADORES	**N = 512**	**N = 1024**	**N = 2048**
T = 1	1.952274	15.931320	127.489117
T = 4	0.576712	4.692198	37.564469
T = 8	0.289309	2.342772	18.798650

Tiempos de ejecución.

	TAMAÑO N		
PROCESADORES	**N = 512**	**N = 1024**	**N = 2048**
T = 4	3.385180125	3.395278716	3.393875127
T = 8	6.748058304	6.8000200788	6.781823003

Cálculos de Speedup

	TAMAÑO N		
PROCESADORES	**N = 512**	**N = 1024**	**N = 2048**
T = 4	0.8462950313	0.848819679	0.8484687818
T = 8	0.8435072955	0.8500025099	0.8477278754

Cálculos de eficiencia.

Nuevamente observamos que los indicadores speedup y eficiencia se aproximan a los ideales, en menor parte que pthreads, pero de todas formas siguen siendo valores aceptables.

02. Búsqueda de minimo y maximo - Pthreads

Paralelizar la búsqueda del mínimo y el máximo valor en un vector de N elementos

En esta ocasión se realizó la paralización de un algoritmo de búsqueda tradicionalmente secuencial, en la cual, generalmente se tiene un arreglo de N-1 posiciones, siendo N la cantidad de datos contenidos y se recorre de 0..N-1 comparando cada valor para obtener la información deseada.

- El algoritmo paralelo realiza una distribución equitativa del *'arreglo[N]'* en T hilos, es decir, cada hilo contará con N/T elementos.
- En un primer momento, cada hilo se encarga de buscar el mínimo y máximo en su porción del vector, al estilo secuencial, y espera por su vecino.
- Posteriormente, la idea es hacer una reducción. Para esto los hilos pasan a trabajar de a pares. Los de ID<T/2 esperan a que sus complementarios (ID+T/2) finalicen, si es que no lo han hecho, y eligen el mínimo/máximo entre ambos.
- En la siguiente etapa, trabajarán la mitad de los hilos y se repite desde el segundo paso.
- Finalmente, en la última etapa el hilo de ID 0 entregará el resultado final.

Para la implementación fue necesario utilizar variables condición y mutex, ambas herramientas de sincronización proveídas por Pthreads. Su utilización parte de la necesidad de evitar condiciones de carreras y mantener una lógica en el orden de ejecución de cada par de procesos interactuantes.

Para la versión secuencial, se ejecutó el algoritmo paralelo con un único hilo, pero contemplando que el main no ejecute *'pthread_create'* y ejecute directamente la función que implementa el comportamiento de los hilos, evitando de esta manera la toma de tiempo de *'pthread_create'* y, obviamente, del *'pthread_join'*.

Los tiempos de ejecución obtenidos fueron los siguientes, utilizando los valores de N que se observan para que los tiempos de ejecución superen el segundo en la mayoría de los casos:

	TAMAÑO N		
PROCESADORES	**N = 500000000**	**N = 1000000000**	**N = 2000000000**
T = 1	3.057723	6.112944	12.219912
T = 4	0.764439	1.529612	3.057116
T = 8	0.385179	0.767107	1.534254

Tiempos de ejecución búsqueda.c

Los resultados obtenidos fueron los esperados ya que al duplicar la cantidad de hilos se logra reducir el tiempo de procesamiento aproximadamente a la mitad.

	TAMAÑO N		
PROCESADORES	**N = 500000000**	**N = 1000000000**	**N = 2000000000**
T = 4	3.999957583	3.996402084	3.997201321
T = 8	7.938428856	7.968819425	7.964721479

Cálculos de Speedup

	TAMAÑO N		
PROCESADORES	**N = 500000000**	**N = 1000000000**	**N = 2000000000**
T = 4	0.999957583	0.999100521	0.9993003303
T = 8	0.992303607	0.9961024281	0.9955901849

Cálculos de Eficiencia

En cuanto Speedup y eficiencia ambos mostrados en las anteriores tablas, observamos valores prácticamente ideales, tanto con 4 u 8 hilos para distintos N.