Año 2024

Sistemas Paralelos

Trabajo Práctico N.º 2 –

Programación en memoria compartida

Grupo 3
AGUSTINA SOL ROJAS – ANTONIO FELIX GLORIOSO CERETTI

Características de hardware y software

Cluster remoto: Cluster Multicore (partición Blade) conformado por 16 nodos. Cada nodo posee 8GB de RAM y 2 procesadores Intel Xeon E5405 de cuatro 4 cores cada uno que operan a 2.0GHz. Compilador gcc (Debian 8.3.0-6) 8.3.0.

Consideraciones respecto a la compilación y ejecución

La compilación de los algoritmos se realizó con el nivel de optimización O3 al igual que en la primer entrega, ya que es el que otorga mejores tiempos de ejecución.

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 cluster remoto. Las prueban 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.

Respuesta

Algoritmo y consideraciones de diseño

En todos los algoritmos, se establece 64 como tamaño del bloque, ya que en la entrega anterior se llegó a la conclusión que era el tamaño más optimo.

Algoritmo secuencial optimizado

Se realizaron algunas modificaciones en el código con respecto a la primer entrega para optimizar el algoritmo secuencial.

Modificaciones realizadas:

1. En el cálculo de los mínimos, máximos y promedios tanto la matriz A como la matriz B se recorren en orden de filas. Esto se realiza de dicha manera para aprovechar la localidad de datos, ya que si la matriz B se recorriera en orden de columnas (fue inicializada de dicha manera) se estaría respetando el "orden" de la estructura pero eso implicaría realizar saltos a ubicaciones distantes en la memoria y no a partes contiguas.
2. En la función "blkmul" se utiliza la variable local "suma" para almacenar las sumas de la posición en un registro en vez de acceder repetidas veces a la memoria principal a buscar los valores que se tienen en el vector "cblk" (hay menor latencia).

Algoritmo paralelo empleando Pthreads

En el código del programa paralelo que emplea Pthreads se realiza lo siguiente:

1. Declaración e implementación de funciones y variables necesarias para el cómputo y el cálculo de tiempo de ejecución.
2. Se define la función worker, que es ejecutada por cada hilo. A cada hilo se le asignara una porción de la matrices sobre la cual trabajar. El hilo calcula los mínimos, máximos y sumas locales de la porción asignada de las matrices A y B. Una vez que se realizan estos cálculos, se utiliza la primitiva mutex para acceder a la sección critica en la cual se comparan los mínimos y máximos locales con los globales y a su vez se realiza la suma para calcular posteriormente los promedios. Una vez que salen de la sección critica los hilos esperan en una barrera necesaria

para el posterior calculo correcto del escalar. Este mismo es calculado por un único hilo (con ID 0) recién cuando todos los hilos hayan alcanzado la barrera. Posteriormente, cada hilo realiza la multiplicación de las matrices A y B y de las matrices C y D para obtener los resultados de sus porciones correspondientes a las matrices resultMatriz y R. Luego, cada hilo espera en una barrera antes de sumar los resultados de las multiplicaciones de sus submatrices y multiplicarlos por el escalar.

Es importante aclarar que cada hilo primero realiza los cálculos locales y luego trabaja con los globales ya que la sección crítica es el cuello de botella en términos de rendimiento. Esto ayuda a minimizar el tiempo que los hilos pasan bloqueados esperando el acceso a la sección critica, permitiendo un mejor aprovechamiento de los recursos.

3. En la función main se verifican los parámetros de entrada. Se aloca memoria para las matrices y se inicializan A, B, C, D con los valores 1.0; R y resultMatriz con los valores 0.0. Luego, se inicializan el mutex y la barrera, y se crean los hilos con “pthread_create”. El programa principal espera a que todos los hilos terminen su trabajo paralelo con la función “pthread_join”. Después se imprime el tiempo de ejecución obtenido, se destruye el mutex y la barrera, y se libera el espacio de memoria de las matrices.

Algoritmo paralelo empleando OpenMP

En el código del programa paralelo que emplea OpenMP se utilizan varias veces las directivas “nowait” para evitar la barrera implícita al final del bucle for, cuando sea necesario, y “schedule(static) para distribuir las iteraciones entre los hilos, asignadolas en forma round-robin.

El algoritmo realiza lo siguiente:

1. Declaración e implementación de funciones y variables necesarias para el cómputo y el cálculo de tiempo de ejecución.
2. Se verifican los parámetros de entrada. Se asigna el número de hilos para OpenMP usando omp_set_num_threads(T). Utilizamos esta función ya que en el archivo .sh, la cantidad de hilos se pasa como parámetro a pesar de que el manual indica que se utilice la linea *export OMP_NUM_THREADS= 8*. Esto se realiza para evitar la

creación de múltiples archivos .sh y dado que no afecta el tiempo de ejecución es una estrategia valida.

3. Se aloca memoria para las matrices y se inicializan A, B, C, D con los valores 1.0; R y resultMatriz con los valores 0.0.
4. Se especifica la región paralela con la directiva “#pragma omp parallel” y se indica que las variables “i, j, k, offsetI, offsetJ, posA, posB” son privadas para cada hilo. Las variables no indicadas son por defecto compartidas.
5. Se realizan los cálculos de mínimos, máximos y sumas de las matrices A y B en paralelo utilizando la directiva “#pragma omp for” con la cláusula “reduction” para evitar interferencias y resultados incorrectos.
6. Un solo hilo calcula los promedios y el escalar usando la directiva “#pragma omp single”.
7. Se realiza la multiplicación de las matrices A con B y C con D en bloques en paralelo utilizando la directiva “#pragma omp for”.
8. Se suma el resultado de las multiplicaciones de matrices y se multiplica por el escalar en paralelo utilizando nuevamente la directiva “#pragma omp for”.
9. Se imprime el tiempo de ejecución obtenido.
10. Se libera el espacio de memoria de las matrices.

Análisis del algoritmo

Tabla con los tiempos de ejecución obtenidos en el cluster remoto al compilar el algoritmo secuencial "matricesCalculo.c" con nivel de optimización O3

N	*512*	*1024*	*2048*	*4096*
Threads				
1	0.454719	3.633988	28.997741	232.068466

Tabla con los tiempos de ejecución obtenidos en el cluster remoto al compilar el algoritmo paralelo que utiliza Pthreads "matricesCalculoPthreads.c" con nivel de optimización O3

N	*512*	*1024*	*2048*	*4096*
Threads				
2	0.228669	1.822285	14.520628	116.079006
4	0.115425	0.919945	7.324422	58.560920
8	0.060109	0.468074	3.726043	29.988328

Tabla con los tiempos de ejecución obtenidos en el cluster remoto al compilar el algoritmo paralelo que utiliza OpenMP "matricesCalculoOpenMP.c" con nivel de optimización O3

N	*512*	*1024*	*2048*	*4096*
Threads				
2	0.213302	1.698398	13.522668	108.247439
4	0.107653	0.857119	6.830799	54.576814
8	0.055042	0.434313	3.451316	27.834653

Gráfico con los tiempos de ejecución obtenidos en el cluster remoto al compilar los algoritmos paralelos con Pthreads y OpenMP con nivel de optimización O3 y N=512

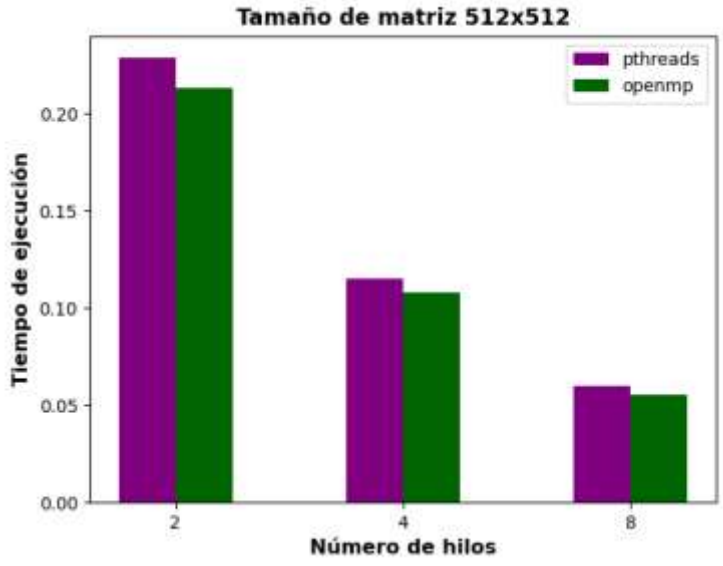


Gráfico con los tiempos de ejecución obtenidos en el cluster remoto al compilar los algoritmos paralelos con Pthreads y OpenMP con nivel de optimización O3 y N=1024

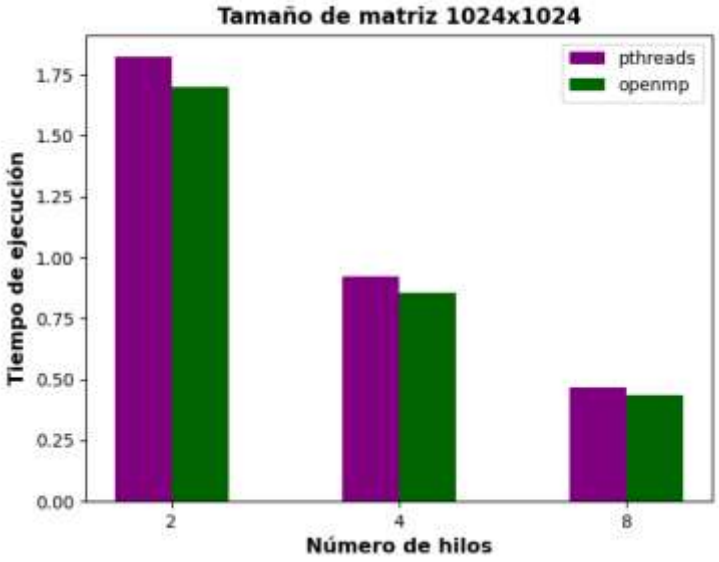


Gráfico con los tiempos de ejecución obtenidos en el cluster remoto al compilar los algoritmos paralelos con Pthreads y OpenMP con nivel de optimización O3 y N=2048

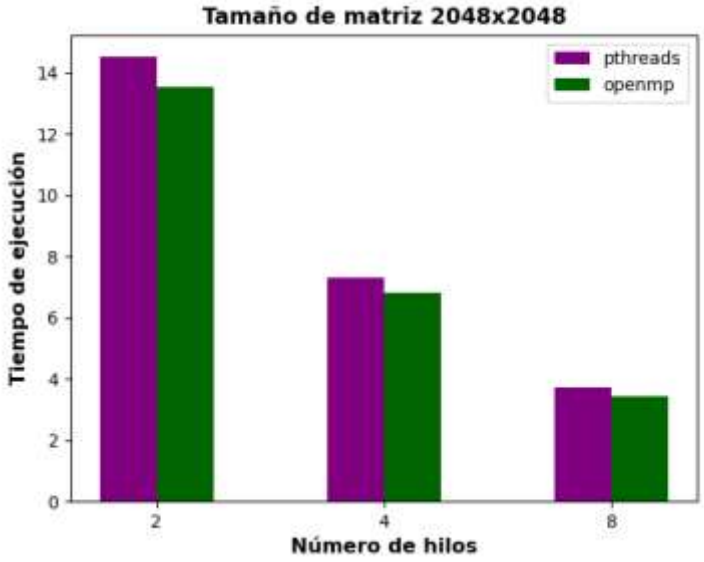


Gráfico con los tiempos de ejecución obtenidos en el cluster remoto al compilar los algoritmos paralelos con Pthreads y OpenMP con nivel de optimización O3 y N=4096

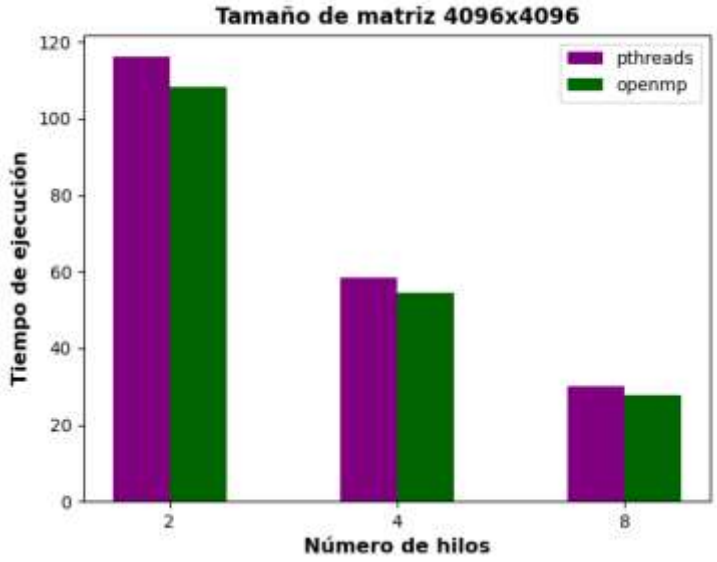
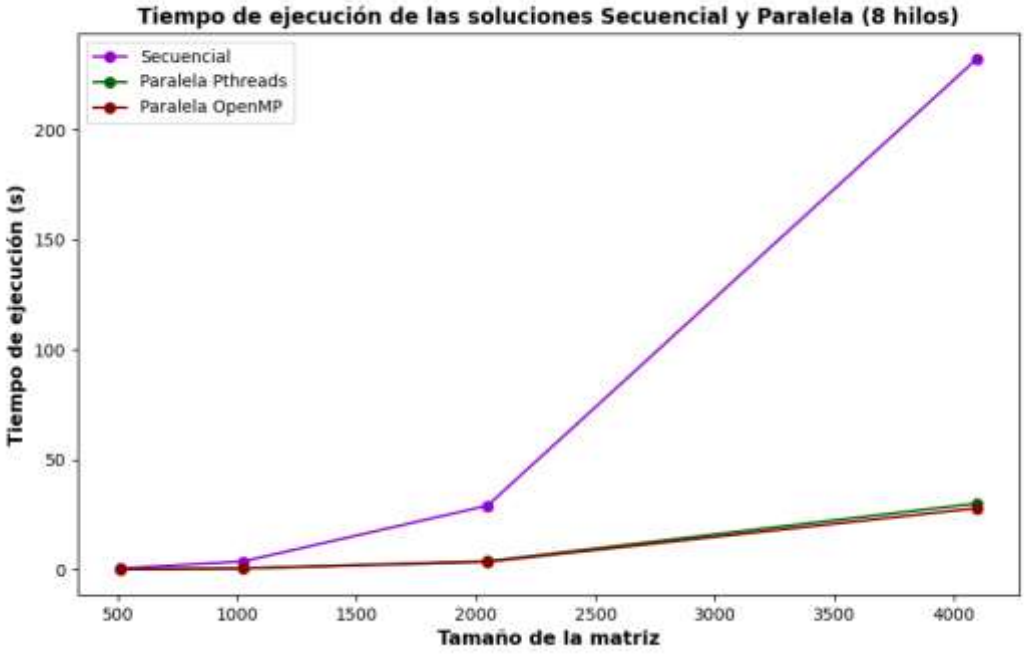


Gráfico con los tiempos de ejecución obtenidos en el cluster remoto al compilar el algoritmo secuencial y los algoritmos paralelos con 8 hilos con nivel de optimización O3



Se puede observar en los primeros gráficos que OpenMP siempre mantiene en todos los N un menor tiempo de ejecución, y la diferencia de tiempo entre los distintos algoritmos siempre se mantiene igual.

Como se puede observar en todas las tablas de algoritmos paralelos, al aumentar el número de hilos (mientras se mantenga menor a la cantidad de unidades de procesamiento, esto se explicará luego) se reduce el tiempo de ejecución. Esta mejoría es similar a un decrecimiento exponencial.

Al comparar las soluciones paralelas con 8 hilos con la secuencial, caso del último gráfico, se nota la gran diferencia de tiempos.

Speedup y Eficiencia

El speedup se calcula de la siguiente manera:

$$Ts / Tp$$

Siendo Tp el tiempo desde que empieza a ejecutarse el primer hilo hasta que termina el último y Ts tiempo que tarda en ejecutarse la solución secuencial. Si el resultado es mayor a 1 es mejor el algoritmo paralelo.

La eficiencia permite saber qué tan cerca se está del speedup óptimo, que tan bien se está aprovechando la arquitectura. Se calcula de la siguiente manera:

$$Speedup/SOptimo$$

El speedup óptimo depende de la arquitectura, si es homogénea es la cantidad de procesadores, en el caso del cluster remoto como se puede ver a través del comando "lscpu" es 8.

El valor de la eficiencia es un número entre 0 y 1, si es más cercano a 1 indica que la arquitectura se está aprovechando, caso contrario es más cercano a 0.

Si el speedup es mayor al speedup óptimo, este será superlineal.

SPEEDUP – PTHREADS

N	*512*	*1024*	*2048*	*4096*
Threads				
2	1.988546764	1.994193005	1.997003229	1.999228577
4	3.939519168	3.950223111	3.959048373	3.962855536
8	7.564907085	7.76370403	7.782449371	7.738626375

Cálculos

N=512

2 hilos 0.454719 / 0.228669 = 1.988546764

4 hilos 0.454719 / 0.115425 = 3.939519168

8 hilos 0.454719 / 0.060109 = 7.564907085

N=1024

2 hilos 3.633988 / 1.822285 = 1.994193005

4 hilos 3.633988 / 0.919945 = 3.950223111

8 hilos 3.633988 / 0.468074 = 7.76370403

N=2048

2 hilos 28.997741 / 14.520628 = 1.997003229

4 hilos 28.997741 / 7.324422 = 3.959048373

8 hilos 28.997741 / 3.726043= 7.782449371

N=4096

2 hilos 232.068466 / 116.079006 = 1.999228577

4 hilos 232.068466 / 58.560920 = 3.962855536

8 hilos 232.068466 / 29.988328 = 7.738626375

SPEEDUP – OPENMP

N	*512*	*1024*	*2048*	*4096*
Threads				
2	2.131808422	2.139656311	2.144380162	2.143870267
4	4.22393245	4.239770674	4.245146285	4.252143887
8	8.261309545	8.367209823	8.401937406	8.337393895

Cálculos

N=512

2 hilos 0.454719 / 0.213302 = 2.131808422

4 hilos 0.454719 / 0.107653 = 4.22393245

8 hilos 0.454719 / 0.055042 = 8.261309545

N=1024

2 hilos 3.633988 / 1.698398 = 2.139656311

4 hilos 3.633988 / 0.857119 = 4.239770674

8 hilos 3.633988 / 0.434313 = 8.367209823

N=2048

2 hilos 28.997741 / 13.522668 = 2.144380162

4 hilos 28.997741 / 6.830799 = 4.245146285

8 hilos 28.997741 / 3.451316 = 8.401937406

N=4096

2 hilos 232.068466 / 108.247439 = 2.143870267

4 hilos 232.068466 / 54.576814 = 4.252143887

8 hilos 232.068466 / 27.834653 = 8.337393895

Se puede observar que en todos los tamaños de N siempre son mejores los algoritmos paralelos y esta mejora es correlativa a la cantidad de hilos, speedup ≈ n.º hilos. Esto probablemente se deba a que se divide la carga computacional entre los hilos, que ejecutan el mismo código de forma paralela sobre diferentes datos.

Se puede ver que en el caso de OpenMP con 8 hilos se tiene un speedup superlineal. Se sabe que este resultado es anormal pero aun así después de varias pruebas y revisiones de algoritmos siguió ocurriendo. Esto podría deberse al efecto caché producido por las

distintas jerarquías de memoria en una computadora. Cuando se incrementa el número de procesadores, se incrementa también el tamaño de las cachés de los mismos. Con un mayor tamaño de caché, todo el conjunto de trabajo o más parte de él puede almacenarse en caché, lo que disminuye considerablemente el tiempo de acceso a la memoria. Esto podría provocar un speedup adicional al producido por el cómputo real.

EFICIENCIA – PTHREADS

N	*512*	*1024*	*2048*	*4096*
Threads				
2	0.994273382	0.9970965025	0.9985016145	0.9996142885
4	0.984879792	0.98755577775	0.98976209325	0.990713884
8	0.9456133856	0.9704630038	0.9728061714	0.9673282969

Cálculos

N=512

2 hilos 1.988546764 / 2 = 0.994273382

4 hilos 3.939519168 / 4 = 0.984879792

8 hilos 7.564907085 / 8 = 0.9456133856

N=1024

2 hilos 1.994193005 / 2 = 0.9970965025

4 hilos 3.950223111 / 4 = 0.98755577775

8 hilos 7.76370403 / 8 = 0.9704630038

N=2048

2 hilos 1.997003229 / 2 = 0.9985016145

4 hilos 3.959048373 / 4 = 0.98976209325

8 hilos 7.782449371 / 8 = 0.9728061714

N=4096

2 hilos 1.999228577 / 2 = 0.9996142885

4 hilos 3.962855536 / 4 = 0.990713884

8 hilos 7.738626375 / 8 = 0.9673282969

EFICIENCIA – OPENMP

N	*512*	*1024*	*2048*	*4096*
Threads				
2	1.065904211	1.0698281555	1.072190081	1.0719351335
4	1.0559831125	1.0599426685	1.06128657125	1.06303597175
8	1.032663693	1.045901228	1.050242176	1.042174237

Cálculos

N=512

2 hilos 2.131808422 / 2 = 1.065904211

4 hilos 4.22393245 / 4 = 1.0559831125

8 hilos 8.261309545 / 8 = 1.032663693

N=1024

2 hilos 2.139656311 / 2 = 1.0698281555

4 hilos 4.239770674 / 4 = 1.0599426685

8 hilos 8.367209823 / 8 = 1.045901228

N=2048

2 hilos 2.144380162 / 2 = 1.072190081

4 hilos 4.245146285 / 4 = 1.06128657125

8 hilos 8.401937406 / 8 = 1.050242176

N=4096

2 hilos 2.143870267 / 2 = 1.0719351335

4 hilos 4.252143887 / 4 = 1.06303597175

8 hilos 8.337393895 / 8 = 1.042174237

Cuando se tiene la misma cantidad de hilos y se aumenta N, la eficiencia mejora. A medida que se tienen más hilos e igual N, la eficiencia decae (no muy significativamente). Tanto Pthreads como OpenMP, en todos los casos, tienen una eficiencia muy cercana al 1. Esto quiere decir que los procesos en ambos están aprovechando casi al 100% el tiempo de CPU. Esto ocurre de dicha manera debido a que la multiplicación de matrices es altamente paralelizable.

En OpenMP se tiene una eficiencia mayor a 1 debido a que antes se obtuvo un speedup superlineal.

Conclusiones

A lo largo del informe se ha hecho notar que utilizar una solución paralela para estos algoritmos de multiplicaciones y cálculos sobre matrices es más eficiente que una secuencial. Mientras más hilos se utilicen y este número se acerque a la cantidad de CPUs de nuestros hardware el tiempo de ejecución se reducirá.

Es recomendable que la cantidad de hilos no supere a la cantidad de unidades de procesamiento, ya que esto implicaría que dos o más hilos se ejecuten en el mismo procesador, implicando un overhead asociado a los cambios de contexto y a la planificación de la CPU, y por lo tanto una sobrecarga relacionada con la concurrencia.