Sistemas Paralelos
Entrega 2
Grupo 12; Juan Andrés Geido

Consideraciones en el testeo:
- Todas las pruebas se hicieron en el cluster a través de Slurm (scripts sbatch)
- Todas las pruebas se hicieron con -O2, ya que en el algoritmo secuencial este presento consistemente el mejor rendimiento.
- Todas las pruebas se hicieron con BS=8, por la misma razón.
- Se incluye un archivo ODS con la plantilla y los cálculos, este PDF los presenta en forma de imagen.

Consideraciones en el desarrollo:
En el caso de ambos algoritmos, el código o “lógica” es mayormente el mismo.

Para el algoritmo pthreads, hice la definición de todas las variables de trabajo en el entorno global, con inicialización en main, y decidí comenzar por separar cada operación (nidos de multiplicaciones y demás cálculos individuales) en pthreads, ya que me pareció la forma mas sencilla de comenzar. Cada hilo calcula su espacio de responsabilidad en base a su id, y cosas como totales y promedios para el escalar se calculan con mutexes entre medio.
Si bien el rendimiento con solo eso no era malo, claramente habían mejoras que hacer.

Luego procedí a mover todo a una sola función (donde ahora se encuentran, “hiloOperacion”), lo cual ahorra andar joineando y creando threads a cada rato. Ademas, le di a cada hilo sus variables locales para llevar lo de las comparaciones y sumas para el escalar por su cuenta, de tal forma que solo se debe utilizar un mutex al final, para comparar los resultados locales con los globales. Estos dos cambios mejoraron el rendimiento dramáticamente. Luego procedí a quitar una barrera innecesaria luego de la multiplicación escalar al darme cuenta que se podía separar la responsabilidad para esa parte de forma similar.
Al final, cada hilo hace su trabajo de forma muy similar al secuencial, con tan solo un mutex y dos barreras, por lo cual debería haber poco overhead por sincronización.

El algoritmo OpenMP termino muy similar al secuencial. En gran parte es simplemente definir el área de calculo como una región parallel, y remarcar en cada for loop las variables privadas para iterar. Se vieron muy pequeños beneficios de usar algunas clausulas: reduction para la medición de los totales en A y B, nowait tras la primera multiplicación ya que esa parte no requiere esperar, cada hilo tiene su responsabilidad, y utilizar el tamaño de bloque como static schedule cuando se hace la multiplicación por el escalar).
Parece tener rendimiento igual o marginalmente mejor a su contraparte Pthreads, por una porción del esfuerzo.

A tener en cuenta:

Si deciden probar el código localmente, vale tener en cuenta: El algoritmo secuencial solo toma dos parámetros (N y BS). El pthreads esos, mas NUM_THREADS. El omp toma opcionalmente un NUM_THREADS, si no lo tomara de donde lo tome por defecto (por ejemplo sbatch).

Las versiones paralelas comparan sus resultados con el secuencial de forma muy rudimentaria: Haciéndolo por segunda vez de forma secuencial. Originalmente lo hice de otra forma mucho menos molesta, en el cual ejecutar la versión secuencial generaba una cache en un .txt para compararla directamente, pero lamentablemente esto no sobrevivía el uso en el cluster a través de sbatch.

Por eso, a las versiones paralelas se les puede indicar un cuarto parámetro, simplemente un “1” al final, para saltear la comprobación y hacer prueba de velocidad mas rápidamente.

Pruebas de rendimiento:

Se muestra con imágenes, pero recordar que el archivo calc original esta incluido.

Velocidad: (Sec 1, Paralelos en 2, 4 y 8)

SECUENCIAL			
NUM_THRD	1	N/A	N/A
N=512	0,56102	N/A	N/A
N=1024	4,60413	N/A	N/A
N=2048	37,00317	N/A	N/A
N=4096	294,13647	N/A	N/A

PTHREADS			
NUM_THRD	2	4	8
N=512	0,28298	0,14203	0,07538
N=1024	2,31696	1,1726	0,59588
N=2048	18,57683	9,42078	4,79027
N=4096	148,66218	74,80227	38,13859

OPENMP			
NUM_THRD	2	4	8
N=512	0,27869	0,14082	0,07246
N=1024	2,23958	1,14142	0,58406
N=2048	17,92245	9,136	4,6734
N=4096	142,87432	72,5602	37,01589

A simple vista se pueden ver resultados no muy sorprendentes, reducciones en el tiempo de ejecución dentro de lo que se esperaría para la cantidad de threads aplicados, con la implementación OpenMP teniendo una ligera pero consistente ventaja sobre la implementación Pthreads.

SPEEDUP

PTHREADS			
NUM_THRD	2	4	8
N=512	1,983	3,950	7,443
N=1024	1,987	3,926	7,727
N=2048	1,992	3,928	7,725
N=4096	1,979	3,932	7,712

OPENMP			
NUM_THRD	2	4	8
N=512	2,013	3,984	7,742
N=1024	2,056	4,034	7,883
N=2048	2,065	4,050	7,918
N=4096	2,059	4,054	7,946

EFICIENCIA

PTHREADS			
NUM_THRD	2	4	8
N=512	0,991	0,988	0,930
N=1024	0,994	0,982	0,966
N=2048	0,996	0,982	0,966
N=4096	0,989	0,983	0,964

OPENMP			
NUM_THRD	2	4	8
N=512	1,007	0,996	0,968
N=1024	1,028	1,008	0,985
N=2048	1,032	1,013	0,990
N=4096	1,029	1,013	0,993

En linea con lo que se vio en velocidad, la implementación OpenMP parece tener una eficiencia consistemente mayor a la implementación en Pthreads.

Observaciones:

- En varios casos OpenMP presenta una eficiencia mayor a 1. Como posible explicación, tal vez OpenMP transparentemente optimiza las operaciones de tal forma que, tener varios núcleos trabajando sobre sus partes del problema, por separado y con sus propias caches dio resultados tan buenos como para superar los overhead de sincronización.
- Cuanto menor el tamaño del problema y mas threads, menor la eficiencia. Mas notablemente el caso de N=512 con 8 threads. El costo de sincronización se vuelve comparativamente mayor como para ignorarlo.
- Aun en los casos de N grande, tirarle mas threads al problema sigue teniendo un efecto negativo en la eficiencia. Aún así, OpenMP se mantiene sobre .99, y pthreads sobre .96, lo cual no parece problemático.