Sistemas Paralelos

Trabajo Práctico 2

Programación

en memoria compartida

Grupo 13

AÑO 2023 – 1° Semestre

Alumnos

- Garofalo, Pedro 17136/5
- Morena, Nahuel 16290/1

Características del equipo hogareño utilizado para las siguientes consignas:

- Procesador: AMD Ryzen 9 5950X (16 núcleos, 32 hilos)
- Memoria: 64Gb DDR4 3.600MHz
- Sistema Operativo: Arch Linux (Linux 6.2.10)
- Compilador: GCC 12.2.1

Dada la siguiente expresión:

$$R = \frac{(MaxA \times MaxB - MinA \times MinB)}{PromA \times PromB} \times [A \times B] + [C \times Pot2(D)]$$

- **Donde A, B, C y R son matrices cuadradas de NxN con elementos de tipo double.**
- **D es una matriz de enteros de NxN y debe ser inicializada con elementos de ese tipo (NO float ni double) en un rango de 1 a 40.**
- **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.**
- **La función Pot2(D) aplica potencia de 2 a cada elemento de la matriz D.**

Desarrolle 3 algoritmos que computen la expresión dada:

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

Los algoritmos deben respetar la expresión dada, es decir no deben realizarse simplificaciones matemáticas.

Los resultados deben validarse comparando la salida del algoritmo secuencial con la salida del algoritmo paralelo.

Mida el tiempo de ejecución de los algoritmos en el cluster 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 el primero la versión secuencial *'mat.c'*, la cual utiliza la misma estructura presentada en la anterior entrega, pero que fue adaptada para complementarse con la solución brindada en esta entrega.

En base a la solución secuencial, se desarrollaron las 2 versiones restantes en donde por un lado se utiliza el estándar Pthreads y por el otro el estándar OpenMP (cuyos archivos fueron llamados *'mat_pthreads.c'* y *'mat_openmp.c'* respectivamente).
Aprovechando que las 3 versiones utilizan porciones de código idénticos, se decidió por implementar el archivo *'matlib.h'* el cual recopila esas porciones para que sean reutilizadas. También se agregó a 'matlib.h' código para inicializar las matrices con la opción de cargarlas desde un archivo y almacenar el resultado en otro archivo, esto nos permitió más adelante validar los resultados obtenidos. Todos estos archivos se encuentran en el directorio *'src'*.

Para automatizar las pruebas, se llevaron a cabo 3 scripts:

- build.sh: Su función es la de compilar los archivos .c si estos son requeridos.
- runtests.sh: Su función es la de evaluar que los resultados obtenidos para la ecuación planteada en cada una de las versiones desarrolladas sean correctas. Para eso, ejecuta cada versión, almacenando en un archivo temporal el resultado obtenido, para posteriormente compararlos con el resultado calculado por el programa '*matrixtool.py*' implementado en Python usando NumPy.
- runtimes.sh: Su función es la de tomar los tiempos que demoran cada una de las 3 versiones, con los distintos tamaños de matrices y cantidad de hilos (en las soluciones paralelas) para calcular la ecuación brindada. Los datos obtenidos serán almacenados en un archivo .csv para su posterior análisis.

Tanto el script para evaluar si la salida es correcta como el script para tomar los tiempos, son completamente independientes, por lo que pueden ser ejecutados individualmente según sea necesario.
Para la multiplicación de matrices por bloque decidimos utilizar un tamaño de submatriz de 64x64 elementos, ya que en la entrega anterior pudimos observar mejores resultados en el clúster remoto.

Resultados obtenidos

Algoritmo secuencial:

	N = 512	N = 1024	N = 2048	N = 4096
T = 1	0,498	3,940	31,416	250,724

Tiempo de ejecución del algoritmo en cluster remoto, en segundos
(N: tamaño de las matrices, T: cantidad de hilos)

Algoritmo paralelo usando pthreads:

	N = 512	N = 1024	N = 2048	N = 4096
T = 2	0,253	1,987	15,711	125,410
T = 4	0,129	1,004	7,920	63,131
T = 8	0,069	0,510	3,997	31,942

Tiempo de ejecución del algoritmo en cluster remoto, en segundos
(N: tamaño de las matrices, T: cantidad de hilos)

Algoritmo paralelo usando OpenMP:

	N = 512	N = 1024	N = 2048	N = 4096
T = 2	0,252	1,981	15,705	125,660
T = 4	0,129	1,000	7,922	62,998
T = 8	0,067	0,510	3,992	31,920

Tiempo de ejecución del algoritmo en cluster remoto, en segundos
(N: tamaño de las matrices, T: cantidad de hilos)

Speedup (pthreads):

	N = 512	N = 1024	N = 2048	N = 4096
T = 2	1,966	1,982	2,000	1,999
T = 4	3,869	3,923	3,967	3,971
T = 8	7,249	7,726	7,860	7,849

Valores de Sp(n) obtenidos con el algoritmo de pthreads
(N: tamaño de las matrices, T: cantidad de hilos)

Speedup (OpenMP):

	N = 512	N = 1024	N = 2048	N = 4096
T = 2	1,973	1,989	2,000	1,995
T = 4	3,872	3,939	3,966	3,980
T = 8	7,389	7,719	7,869	7,855

Valores de Sp(n) obtenidos con el algoritmo de OpenMP
(N: tamaño de las matrices, T: cantidad de hilos)

Eficiencia (pthreads):

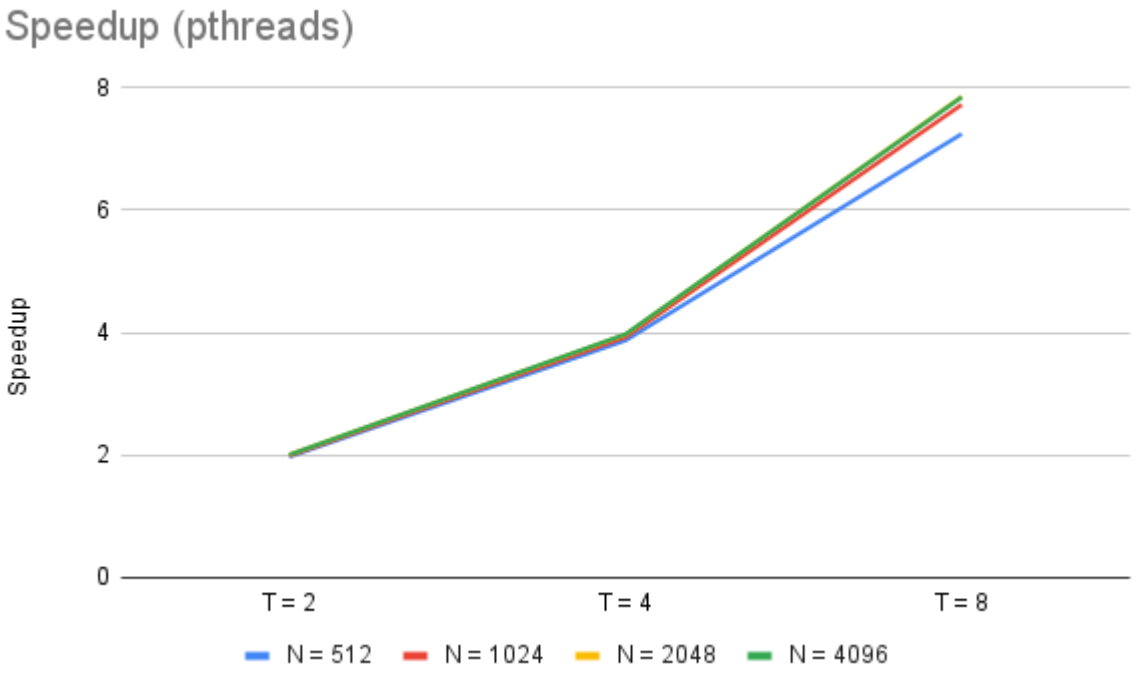
	N = 512	N = 1024	N = 2048	N = 4096
T = 2	0,983	0,991	1,000	1,000
T = 4	0,967	0,981	0,992	0,993
T = 8	0,906	0,966	0,983	0,981

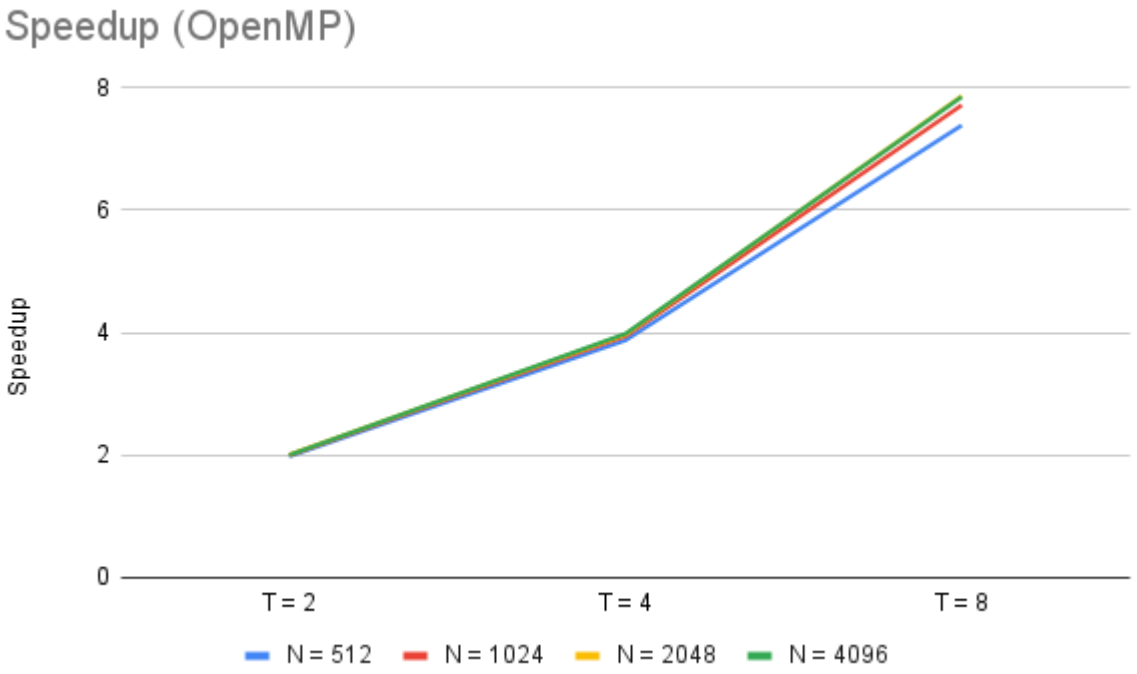
Valores de Ep(n) obtenidos con el algoritmo de pthreads
(N: tamaño de las matrices, T: cantidad de hilos)

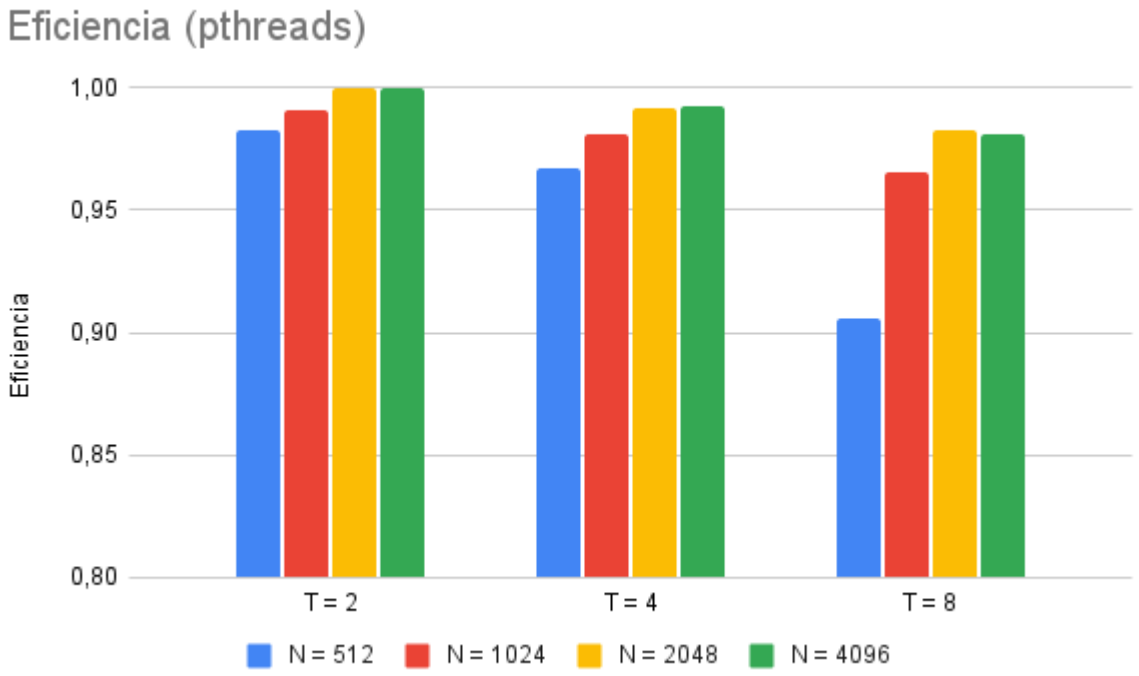
Eficiencia (OpenMP):

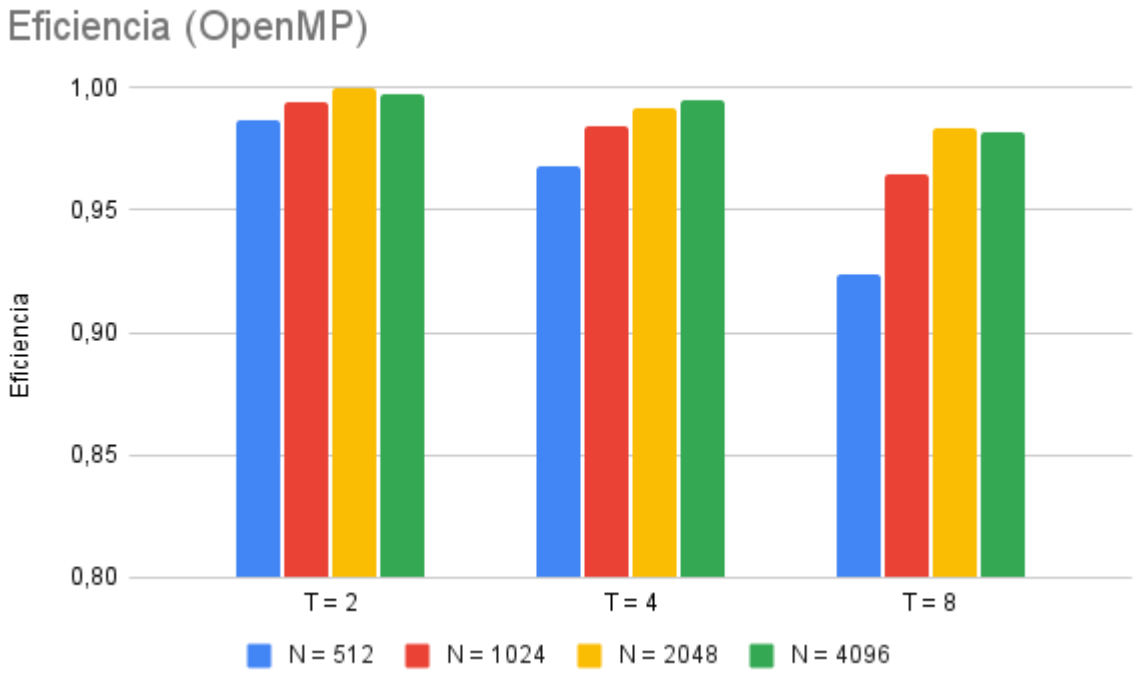
	N = 512	N = 1024	N = 2048	N = 4096
T = 2	0,987	0,995	1,000	0,998
T = 4	0,968	0,985	0,991	0,995
T = 8	0,924	0,965	0,984	0,982

Valores de Ep(n) obtenidos con el algoritmo de OpenMP
(N: tamaño de las matrices, T: cantidad de hilos)


Speedup (pthreads)
Speedup
8
6
4
2
0
T = 2
T = 4
T = 8
N = 512
N = 1024
N = 2048
N = 4096


Speedup (OpenMP)
Speedup
8
6
4
2
0
T = 2
T = 4
T = 8
N = 512
N = 1024
N = 2048
N = 4096


Eficiencia (pthreads)
1,00
0,95
0,90
0,85
0,80
Eficiencia
T = 2
T = 4
T = 8
N = 512
N = 1024
N = 2048
N = 4096


Eficiencia (OpenMP)
1,00
0,95
0,90
0,85
0,80
Eficiencia
T = 2
T = 4
T = 8
N = 512
N = 1024
N = 2048
N = 4096

Observando las tablas y gráficos anteriores podemos llegar a la conclusión que ambos algoritmos paralelos tienen un rendimiento muy similar.
También podemos ver que la eficiencia mejora cuando el tamaño de las matrices aumenta, y que empeora cuanto más hilos son utilizados.
Podemos observar que la eficiencia es cercana a 1 en todos los algoritmos y casos probados, esto se debe a que la multiplicación de matrices es altamente paralelizable.