Sistemas Paralelos
Entrega 2

Integrantes:

13926/1 Cazorla, Ignacio Nicolás
13655/7 Delle Donne, Matías Adrián

Dada la siguiente expresión:

$$R = \frac{Max\_F.Min\_F}{Prom\_F}.[A.B.C + D.Fib(F)]$$

− Donde A, B, C, D y R son matrices cuadradas de NxN con elementos de tipo double.
− F 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.
− Max_F y Min_F son los valores máximo y mínimo de la matriz F, respectivamente.
− Prom_F es el valor promedio de los elementos de la matriz F.
− La función Fib(F) aplica Fibonacci a cada elemento de la matriz F.

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 del algoritmo secuencial con la salida del algoritmo paralelo. Posiblemente deban tener en cuenta algún grado de error debido a la precisión en el cálculo.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.

1. Algoritmo secuencial optimizado

Con un análisis previo[1] del algoritmo secuencial para resolver la expresión, se determinó que:

- La mejor opción para la multiplicación de matrices fue el algoritmo de multiplicación por bloques, ya que fue el que mayor rendimiento proporcionó.

- Para un tamaño de matriz NxN con N = 1024, los mejores tamaños para la multiplicación por bloques son 32, 64, 128 o 256.

- Con alguno de los tamaños de bloque indicados, se ejecutaron los programas con optimizaciones y los resultados obtenidos fueron:

BS	Sin opt.	Opt. -O1	Opt. -O2	Opt. -O3
32	39.857428	11.588985	5.917513	6.743961
64	39.299601	13.156754	5.715443	6.366578
128	38.786574	15.083599	6.015808	6.458808
256	38.590304	14.925730	5.833189	6.301476

- Finalmente se eligen los tamaños de bloque 32 y 64 con la optimización 2 (-O2), para ejecutar este programa secuencial con tamaños de entrada 512, 1024, 2048 y 4096.

2. Algoritmo paralelo empleando Pthreads

Explicación

Para lograr que el algoritmo secuencial sea paralelo, se introdujeron cambios estructurales en el código, principalmente en la función **main()**.
Todas las operaciones relacionadas a resolver la expresión se llevan a cabo en la función **realizar_calculo()**. La gran mayoría de las variables que contenía **main()** ahora pasan a ser globales, para simplificar el acceso a la información.

[1] Informe sobre la entrega 1.

Compilación

```
$ gcc -pthread -o calculo_pthread calculo_pthreads.c mmblk_pthread.c -O2
```

Optimización

Desde **main()** se crea un conjunto de hilos que resolverán el problema dividiendo cada etapa del cálculo en partes más pequeñas. Es decir, para el tamaño del problema indicado, cada hilo será responsable de calcular una porción de ese problema.

Para lograr este objetivo hay algunos puntos en los que es necesario la sincronización, para ello se utilizaron barreras. Respecto del código secuencial, fue necesario reorganizar instrucciones para poder obtener el máximo rendimiento. Por ejemplo, una vez que se calcula el arreglo con los valores de Fibonacci entre 1 y 40, seguidamente se realiza la multiplicación de las matrices A y B, y luego AB y C. De esta manera podemos realizar trabajo útil mientras no esté listo el arreglo con los valores.
Fue necesario establecer tres barreras que establecen dependencias entre las etapas.

También, utilizamos un mutex para la actualización de valores mínimo, máximo y una suma común a todos los hilos. Para esto, se decidió obtener cada uno de estos valores de manera local a cada hilo y finalmente realizar una actualización con un mutex al final de cada ejecución.

Ejecuciones

N	BS	Tiempo secuencial	Tiempo paralelo	Speedup	Eficiencia
512	64	0,716878	0,364007	1,969407182	0,984703591
1024	64	5,715443	2,900182	1,970718734	0,985359367
2048	64	45,611940	22,986870	1,98426058	0,99213029
4096	64	372,316681	180,213007	2,065981181	1,03299059

2 hilos

N	BS	Tiempo secuencial	Tiempo paralelo	Speedup	Eficiencia
512	64	0,716878	0,187513	3,823084266	0,955771067
1024	64	5,715443	1,494052	3,825464576	0,956366144
2048	64	45,611940	11,652769	3,914257633	0,978564408
4096	64	372,316681	90,964403	4,092993179	1,023248295

4 hilos

N	BS	Tiempo secuencial	Tiempo paralelo	Speedup	Eficiencia
512	64	0,716878	0,104325	6,871583992	0,858947999
1024	64	5,715443	0,774455	7,379954936	0,922494367
2048	64	45,611940	5,940648	7,677940184	0,959742523
4096	64	372,316681	46,772974	7,960081414	0,995010177

8 hilos

3. Algoritmo paralelo empleando OpenMP

Explicación

Se utiliza el mismo método que en ***Pthread***, es decir, el algoritmo paralelo consiste en dividir las iteraciones en partes iguales para resolver la ecuación en un menor tiempo y se aprovecha la división de las matrices para que los *threads* no esperen que termine una parte para realizar la siguiente.

Seteamos la cantidad de hilos enviando como parámetro para la función **omp_set_num_threads()**.

Utilizamos el constructor *parallel* que nos permite especificar un bloque de código será ejecutado en paralelo, la cláusula *private* para las variables en las iteraciones del ***for*** como i, j y k, creando una copia para cada hilo. Por último, aplicamos las reducciones de suma, max y min para la matriz de Fibonacci, utilizando la cláusula *reduction*.

Compilación

```
$ gcc -fopenmp -o calculo_openmp openmp.c mmblk.c -O2
```

Ejecuciones

N	BS	Tiempo secuencial	Tiempo paralelo	Speedup	Eficiencia
512	64	0,716878	0,363812	1,970462766	0,985231383
1024	64	5,715443	2,876559	1,986902754	0,993451377
2048	64	45,611940	23,031493	1,98041612	0,99020806
4096	64	372,316681	180,428448	2,063514291	1,031757146

2 hilos

N	BS	Tiempo secuencial	Tiempo paralelo	Speedup	Eficiencia
512	64	0,716878	0,186157	3,850932278	0,962733069
1024	64	5,715443	1,491937	3,830887631	0,957721908
2048	64	45,611940	11,625707	3,923369134	0,980842283
4096	64	372,316681	90,934369	4,094345022	1,023586255

4 hilos

N	BS	Tiempo secuencial	Tiempo paralelo	Speedup	Eficiencia
512	64	0,716878	0,102091	7,021951004	0,877743876
1024	64	5,715443	0,774743	7,37721154	0,922151442
2048	64	45,611940	5,954763	7,659740614	0,957467577
4096	64	372,316681	46,761760	7,961990331	0,995248791

8 hilos