**Entrega 2**

Alumnos: Franco Montanari y Ramiro Antonio

Estrategia de paralelización:

Pensamos un algoritmo en el que cada thread realice la búsqueda del máximo, mínimo y suma de todos los elementos de ¼ de cada una de las matrices a multiplicar y una vez que ese thread saco estos números buscados, actualiza los máximo, mínimo y suma totales al mismo tiempo que realiza la multiplicación de todas las matrices (solo de la parte que le tocó). Una vez que cada thread termina de realizar sus cálculos, se debe esperar a que todos los demás threads terminen de realizar sus tareas, y luego se calcula el escalar pedido y se vuelve a indicar a cada thread que realice la multiplicación de su parte de la matriz resultado por el escalar. Por lo tanto la estrategia utilizada es una paralelización de datos de entrada y salida ya que se toman en cuenta las matrices de entrada para dividir las tareas para cada uno de los hilos y al finalizar cada uno procesa su parte de la matriz resultado.

Algoritmo secuencial:

|  |  |  |
| --- | --- | --- |
| M / N | 512 | 1024 |
| 3 | 1,512 s | 12,412 s |
| 4 | 2,323 s | 18,826 s |
| 5 | 3,110 s | 25,260 s |
| 6 | 3,875 s | 31,733 s |
| 7 | 4,655 s | 37,626 s |
| 8 | 5,473 s | 44,366 s |

Algoritmo paralelo en Pthreads:

|  |  |  |
| --- | --- | --- |
| M / N | 512 | 1024 |
| 3 | 0,444 s | 3,381 s |
| 4 | 0,653 s | 5,108 s |
| 5 | 0,866 s | 6,805 s |
| 6 | 1,072 s | 8,535 s |
| 7 | 1,286 s | 10,263 s |
| 8 | 1,498 s | 11,968 s |

Algoritmo paralelo en OpenMP:

|  |  |  |
| --- | --- | --- |
| M / N | 512 | 1024 |
| 3 | 0,443 s | 3,368 s |
| 4 | 0,653 s | 5,072 s |
| 5 | 0,867 s | 6,732 s |
| 6 | 1,072 s | 8,544 s |
| 7 | 1,287 s | 10,364 s |
| 8 | 1,502 s | 12,006 s |

Speedup Pthreads

|  |  |  |
| --- | --- | --- |
| M / N | 512 | 1024 |
| 3 | 3,405 | 3,671 |
| 4 | 3,557 | 3,685 |
| 5 | 3,591 | 3,711 |
| 6 | 3,614 | 3,717 |
| 7 | 3,619 | 3,666 |
| 8 | 3,653 | 3,707 |

Speedup OpenMP

|  |  |  |
| --- | --- | --- |
| M / N | 512 | 1024 |
| 3 | 3,413 | 3,685 |
| 4 | 3,557 | 3,711 |
| 5 | 3,587 | 3,752 |
| 6 | 3,614 | 3,714 |
| 7 | 3,616 | 3,633 |
| 8 | 3,643 | 3,695 |

Eficiencia Pthreads

|  |  |  |
| --- | --- | --- |
| M / N | 512 | 1024 |
| 3 | 0,851 | 0,917 |
| 4 | 0,889 | 0,921 |
| 5 | 0,897 | 0,927 |
| 6 | 0,903 | 0,929 |
| 7 | 0,840 | 0,916 |
| 8 | 0,913 | 0,942 |

Eficiencia OpenMP

|  |  |  |
| --- | --- | --- |
| M / N | 512 | 1024 |
| 3 | 0,853 | 0,921 |
| 4 | 0,889 | 0,927 |
| 5 | 0,896 | 0,938 |
| 6 | 0,903 | 0,928 |
| 7 | 0,904 | 0,908 |
| 8 | 0,910 | 0,923 |

Los datos anteriores fueron tomadas en la sala de PC de Posgrado (máquinas provistas por la cátedra).

Conclusiones:

Con Pthreads la paralelización fue más compleja que con OpenMP ya que tuvimos que pensar más en detalle cómo realizar todo el algoritmo, y con OpenMP algunos temas como por ejemplo el cálculo de cuánta información debía procesar cada thread ya se hacía automáticamente con la directiva correspondiente. El código con OpenMP queda más ordenado y legible ya que las directivas ayudan a entender cómo funciona cada parte del código sin tener que estar viendo el resto del programa.

Con estos algoritmos se pudo obtener un speedup cercano a 3,7 y una eficiencia de 0,9 por lo que llegamos a la conclusión de que el problema provisto es altamente paralelizable con respecto a los datos.