Ejercicio práctico III

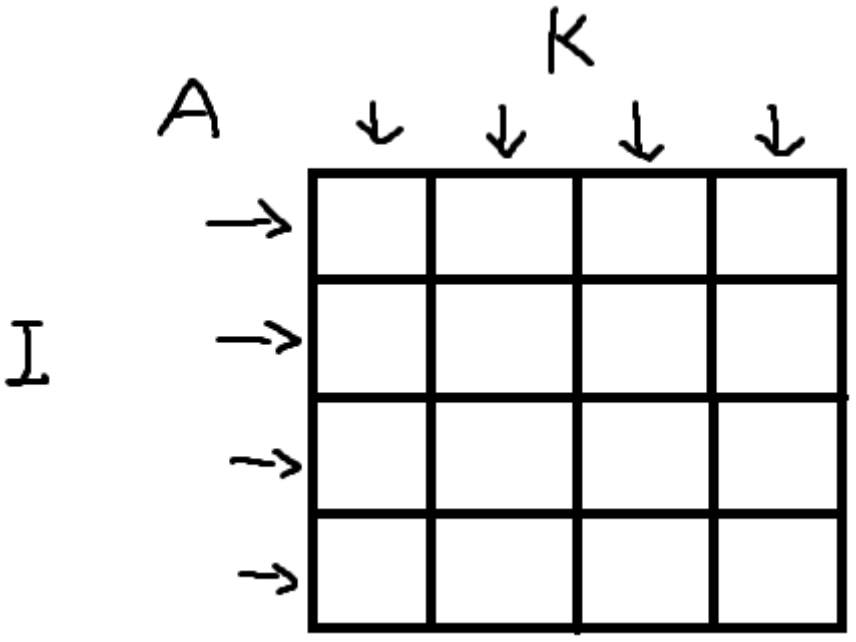
Trabajo realizado por: Antonio Gómez Giménez

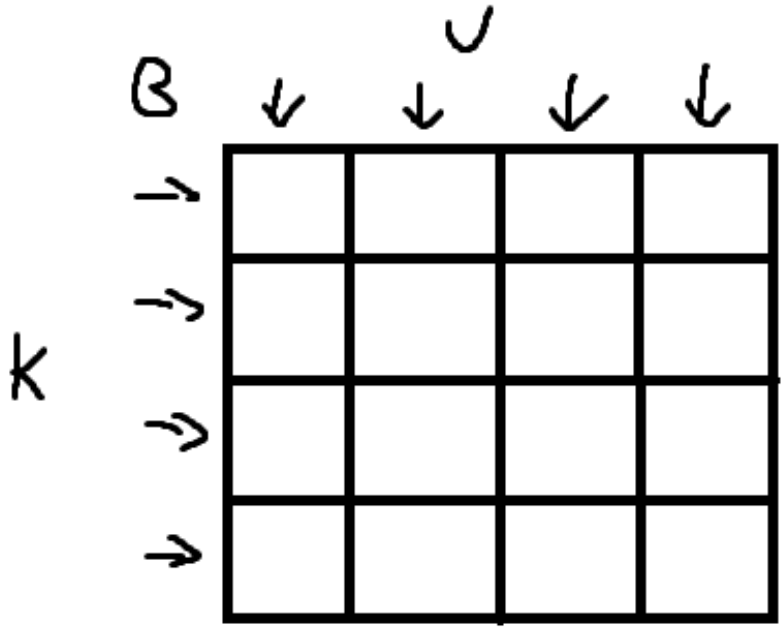
Ejercicios:

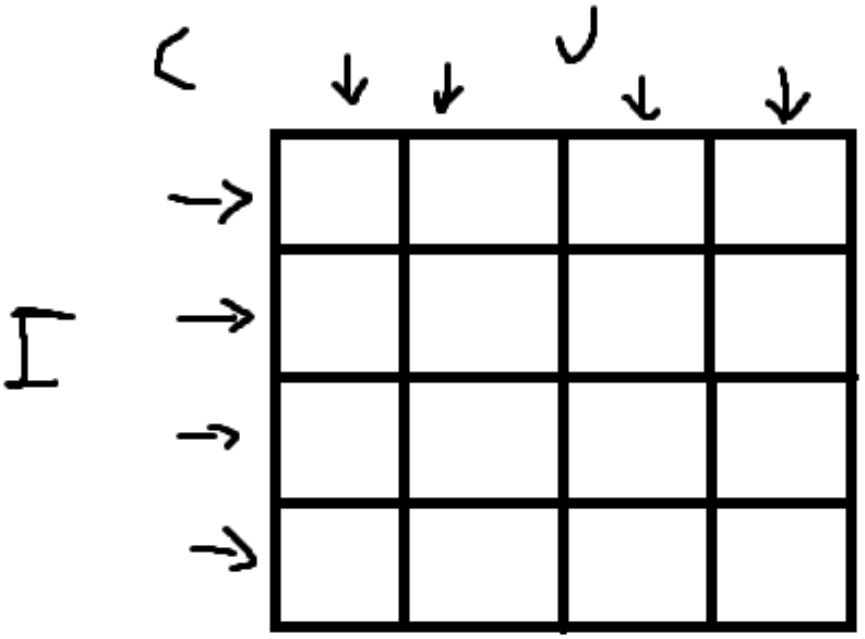
En esta práctica, se busca paralelizar distintos tipos de bucles de iteraciones para el problema visto en prácticas anteriores, la multiplicación de matrices, por tanto, vamos a refrescar tanto la fórmula de multiplicación de matrices, como la forma que tienen las matrices a la hora de paralelizarlas.

La fórmula a paralelizar es la siguiente : C[i][j]= A[i][k]*B[k][j];

La estructura vista en las matrices es la siguiente:







Una vez refrescado esto, vamos a realizar paralelización en tres tipos de bucles ijk, jki e ikj, aplicando la paralelización en el bucle interno como en el bucle interno, una vez comprobemos que paralelización da mejores resultados, realizaremos la paralelización especificando el número de hilos para así comprobar el número óptimo de hilos para esa paralelización.

Por último, se ejecutará el programa secuencial para ver si realmente la paralelización supone una mejora y se propondrá algún tipo de mejora para estos tipos de paralelización.

Cabe destacar que las matrices usadas son del tamaño de 4000x4000 como vimos en la práctica anterior.

1.-Realizar una implementación paralela (con número de hilos por defecto) con vector de iteraciones (i, j, k) paralelizando: a) el bucle más externo y b) el bucle más interno. Con los resultados de tiempo obtenidos desarrollar unas conclusiones.

1.1.-Con la opción del mejor tiempo, realizar pruebas contemplando distinto número de hilos (2, 4, 8, 16, 32, 64) y discutir resultados en base a speed-up y eficiencia.

Tras realizar las pruebas paralelizando únicamente i, como paralelizando únicamente k, los tiempos obtenidos, independientemente de los hilos son:

Paralelización de i: 400.462762

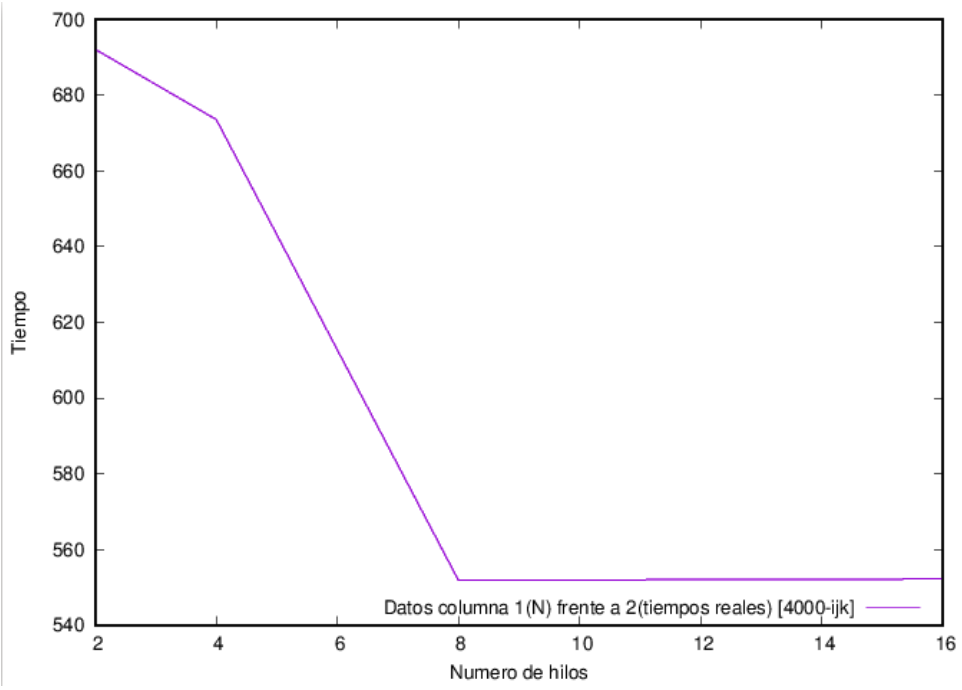
Paralelización de k: 438.023429

Por tanto, es más eficiente la paralelización del iterador i ya que su tiempo es inferior, esto se debe a que paralelizamos la matriz c que es la de mayor importancia porque se realizan más escrituras en la misma, por tanto, dividir esta matriz permite aumentar la eficiencia del programa. Paralelizar k se notaría pero sería únicamente en las matrices A y B, siendo estas utilizadas en la lectura, por ello, la paralelización se notaría menos que en el caso de i.

Una vez que sabemos que la paralelización de bucle i es más eficiente, realizamos las pruebas para 2, 4, 8 y 16, dando los siguientes resultados:

-2 hilos: 692.169087
-4 hilos: 673.620959
-8 hilos: 551.842739
-16 hilos: 552.199566

La gráfica obtenida es la siguiente:



Como podemos observar, los tiempos obtenidos especificando los hilos son peores, esto se debe a que omp si no especificamos los hilos, coje por defecto la cantidad óptima de hilos para paralelizar respecto al iterador especificado, probablemente, el valor óptimo que ha escogido es un valor entre 8 y 16 ya que a partir de 16 la curva empieza a ascender de tal forma que no renta añadir más hilos porque la carga que añade la paralelización empeora el tiempo.

2.-Realizar una implementación paralela (con número de hilos por defecto) con vector de iteraciones (j, k, i) paralelizando: a) el bucle más externo y b) el bucle más interno. Con los resultados de tiempo obtenidos desarrollar unas conclusiones.

2.1.-Con la opción del mejor tiempo, realizar pruebas contemplando distinto número de hilos (2, 4, 8, 16, 32, 64) y discutir resultados en base a speed-up y eficiencia.

Tras realizar las pruebas paralelizando únicamente j, como paralelizando únicamente i, los tiempos obtenidos, independientemente de los hilos son:

Paralelización de j: 1910.095665
Paralelización de i: 602.741947

Cualquiera de las dos métodos de paralelización da tiempos muy peores al resto de las paralelizaciones vistas. Esto se debe a que el tiempo en serie es muy malo, y aunque paralelicemos los problemas de caché van a seguir existiendo. Para que se entienda vamos a ver qué está pasando en serie como vimos en la práctica 1.

La fórmula es la siguiente: C[i][j]= A[i][k]*B[k][j];
Si recordamos en la práctica 1, ijk era peor que ikj, esto se debía a que en la matriz b había constantes fallos de memoria y esto se podía evitar en ikj, ya que se evitaban muchos casos de fallo de caché, esto está explicado mejor en la práctica 1. A lo que quiero llegar es que a la hora de iterar los bucles buscamos que el bucle más interno pertenezca a la segunda dimensión de la matriz y que el bucle más externo pertenezca a la primera dimensión, de esta forma evitaremos los fallos de caché. Por tanto, si usamos jki, la matriz B va a dar constantemente fallos de cache, pero no solo eso, sino que la matriz C también va a tener constantemente fallos de caché y ocurre lo mismo con la matriz A, por tanto, todas las matrices van a dar constantemente dando fallos de caché ralentizando la eficiencia del programa. Por ello, si el programa es lento en serie, en paralelo se mejorará el tiempo pero será más lento que en otras paralelizaciones sobre programas en serie más eficientes.

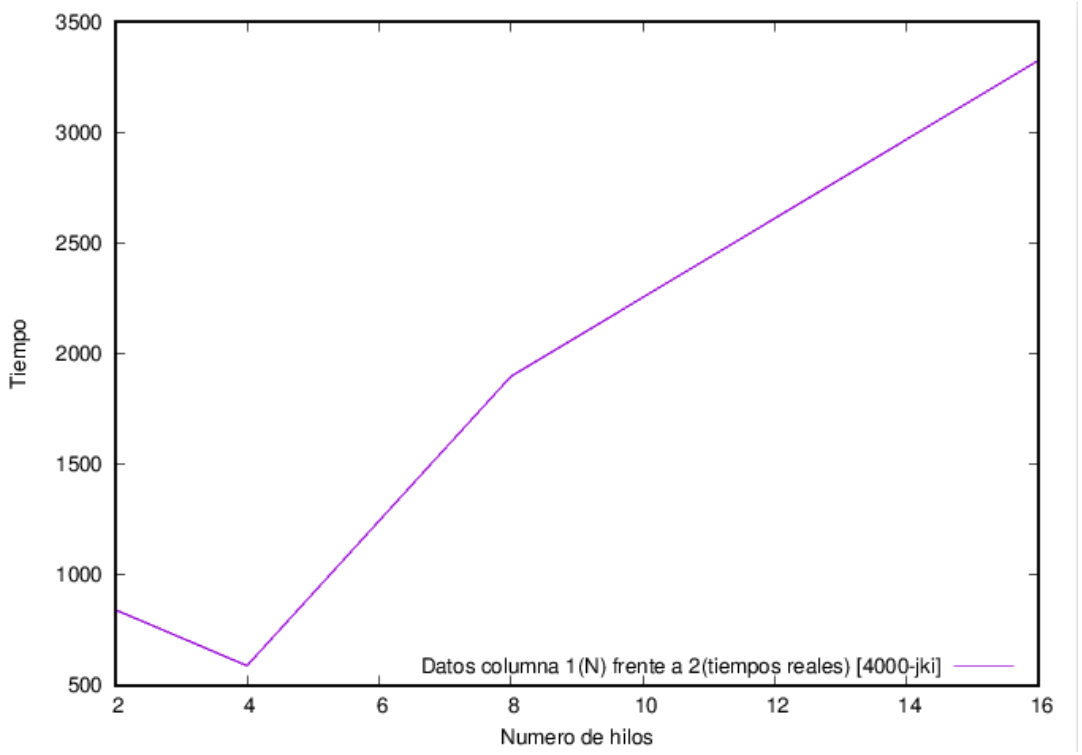
Respecto a los dos tipos de paralelización i es más eficiente respecto a j porque si paralelizamos con i cogemos filas mientras que si paralizamos con j cogemos columnas, a lo que nos referimos es que j, al paralelizar tendría que realizar C[0][0], C[1][0], C[2][0] y asi hasta el final de la matriz, en este caso 4000, si suponemos que j se paraleliza en 4000 hilos, cada hilo tendría una columna como estas y como vimos en la práctica 1 y como he explicado justo en el párrafo anterior, parece que estamos estuviéramos iterando i como si

fuera el bucle más interno, esto lo que genera es que cada vez que se realiza un C en ese hilo al realizar el siguiente C de fallo de caché dando continuamente fallos de cache.
Esto en el caso de i, no ocurre porque sería al contrario, de esta forma se evitan tantos fallos de caché.

Una vez que sabemos que la paralelización de bucle i es más eficiente, realizamos las pruebas para 2, 4, 8 y 16, dando los siguientes resultados:

-2 hilos: 842.029236
-4 hilos: 588.591634
-8 hilos: 1896.658553
-16 hilos: 3326.716443

La gráfica obtenida es la siguiente:



Este caso explica muy bien la eficiencia en proporción a los hilos, como vimos y explicamos en ocasiones anteriores podemos observar cómo incrementar el número de hilos no tiene por qué aumentar la eficiencia, esto se debe a que añadimos mayor carga de trabajo y si el tamaño de la matriz no genera un trabajo útil necesario, se verá reflejada de mayor manera el trabajo extra que genera la paralelización ya que no será proporcional a trabajo útil.
Para este caso, el número óptimo de hilos parece ser 4, de hecho da un mejor resultado que omp cuando realiza la paralelización sin especificarle el número de hilos.

3.-Realizar una implementación paralela (con número de hilos por defecto) con vector de iteraciones (i, k, j) paralelizando: a) el bucle más

externo y b) el bucle más interno. Con los resultados de tiempo obtenidos desarrollar unas conclusiones.

3.1.-Con la opción del mejor tiempo, realizar pruebas contemplando distinto número de hilos (2, 4, 8, 16, 32, 64) y discutir resultados en base a speed-up y eficiencia.

Tras realizar las pruebas paralelizando únicamente i, como paralelizando únicamente j, los tiempos obtenidos, independientemente de los hilos son:

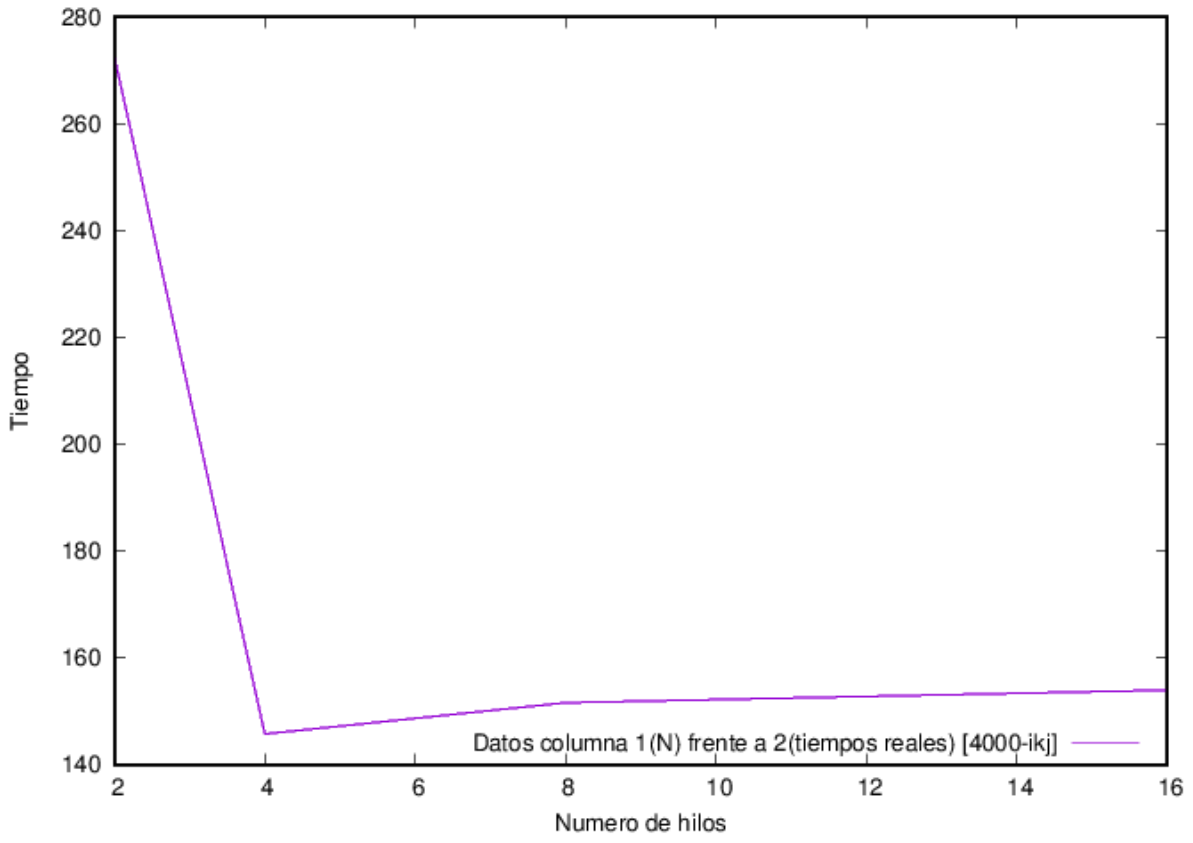
Paralelización de i: 132.572400
Paralelización de j: 190.886303

I es más eficiente que j, la explicación sería la misma que vimos en el caso de jki.

Una vez que sabemos que la paralelización de bucle i es más eficiente, realizamos las pruebas para 2, 4, 8 y 16, dando los siguientes resultados:

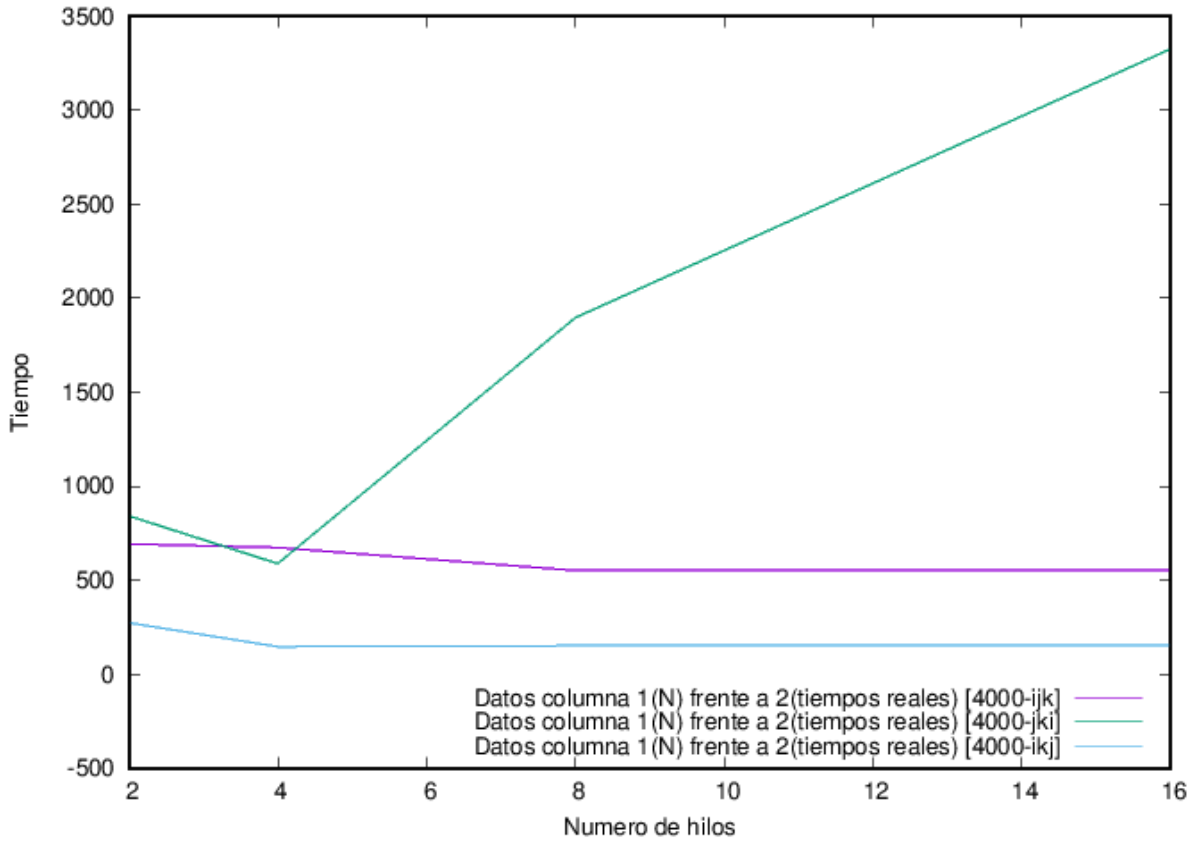
-2 hilos: 272.414212
-4 hilos: 145.713793
-8 hilos: 151.565951
-16 hilos: 153.939680

La gráfica obtenida es la siguiente:



Esta gráfica es similar al caso anterior, podemos observar que el número de hilos más eficiente para el uso de matrices de 4000x4000 viene a ser de 4 hilos (puede ser que 3 sea el óptimo ya omp por defecto da un resultado un poco mejor que el mejor visto en esta gráfica), todos los valores de números de hilos superior a 4, para este caso, van a tener un rendimiento menor por lo explicado en el apartado anterior.

Como curiosidad se muestra en una gráfica conjunta todas las gráficas anteriores para así observar cual de todos los métodos, aplicando todos estos hilos, nos da mejores resultados:



Como era de esperarse, la paralelización de ikj es la más eficiente, esta gráfica es un reflejo de los tiempos en serie ya que si paralelizas algo con buen tiempo te dará buen tiempo, si paralelizas algo con peor tiempo el resultado será una mejora respecto a ese tiempo peor.
A lo que me refiero es que ijk en serie es el más rápido, por ello al paralelizar da mejores tiempos, jki da peores tiempos, por tanto la paralelización permite mejorar tiempos pero no lo suficiente para mejorar los tiempos de un paralelo que en serie tenía mejores tiempos.
Los tiempos en serie lo vemos en el siguiente apartado.

4.-Realizar la ejecución secuencial del programa con el mismo número y vectores de iteraciones anteriores y comparar resultados de dicha

ejecución secuencial con las mejores opciones de paralelización de los 3 puntos anteriores.

Datos en serie:

Datos de bucle ijk en serie para matrices de 4000 valores:
957.320773
Datos de bucle jki en serie para matrices de 4000 valores:
2975.602198
Datos de bucle ikj en serie para matrices de 4000 valores:
407.826484

Datos obtenidos en esta práctica (mejores casos):

Datos de bucle ijk en paralelo (8 hilos) para matrices de 4000 valores:
551.842739
Datos de bucle jki en paralelo (4 hilos) para matrices de 4000 valores:
588.591634
Datos de bucle ikj en paralelo (4 hilos) para matrices de 4000 valores:
145.713793

Como podemos ver y como comentamos anteriormente, todos los casos de paralelización dan mejores resultados frente a ejecución en serie. Cabe destacar que el hecho de que estemos utilizando matrices de 4000 beneficia a los programas en serie ya que evitamos que se machaque la misma posición (línea) en al memoria caché como vimos en la práctica 1, de hecho, los tiempos obtenidos se podrían mejorar con matrices de 4096 como vimos en la práctica 2, ya que en las fronteras evitaríamos que se compartan bloques de la caché en distintos hilos.

Cabe destacar que el tiempo dado por jki en serie es peor que el resto por lo comentado en el apartado 1 y como vimos en el resto de apartados esto afecta a la paralelización.

5.-¿Aplicarías mejoras en alguna de las implementaciones paralelas anteriores para obtener mejores tiempos? ¿Qué mejora y dónde?

Una mejora clara que se puede aplicar a cualquiera de las parelizaciones realizadas ya sea ijk, jki o ikj, dando igual el iterador a paralelizar, es el hecho de cambiar el tamaño de las matrices a 4096, ya que como vimos en la práctica anterior, así evitamos que varios hilos compartan bloque de cache los cuales deben de actualizar cada vez que reciben modificaciones.

Otra mejora que se puede realizar es por ejemplo en el caso de ijk, como vimos en la práctica anterior, se puede paralelizar tanto i como j, por tanto, si tenemos en presente el

hecho de cómo paralelizamos, es una mejora a tener en cuenta. A lo que me refiero, es que habría que especificar a openmp el número de hilos a usar en cada bucle for.