Ejercicio práctico II

Trabajo realizado por: Antonio Gómez Giménez

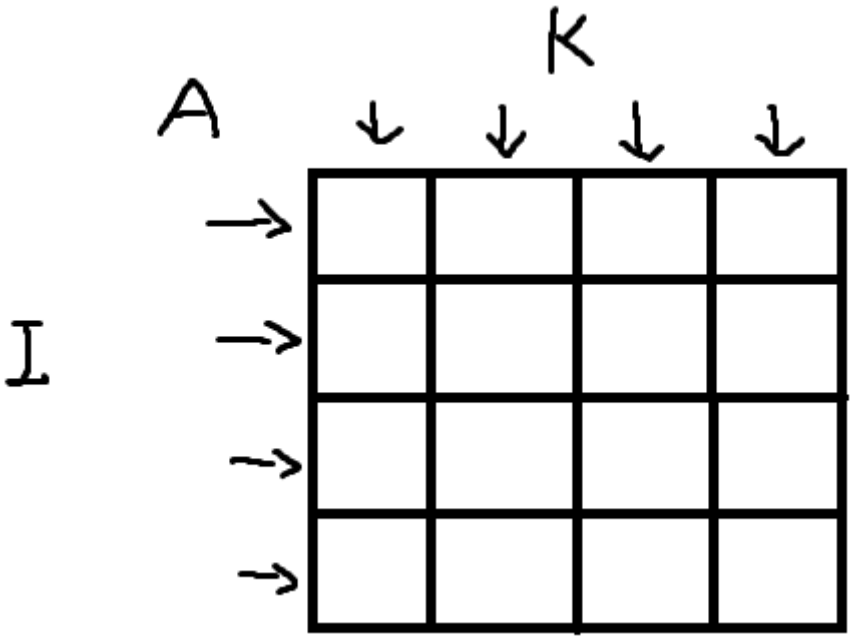
Ejercicios:

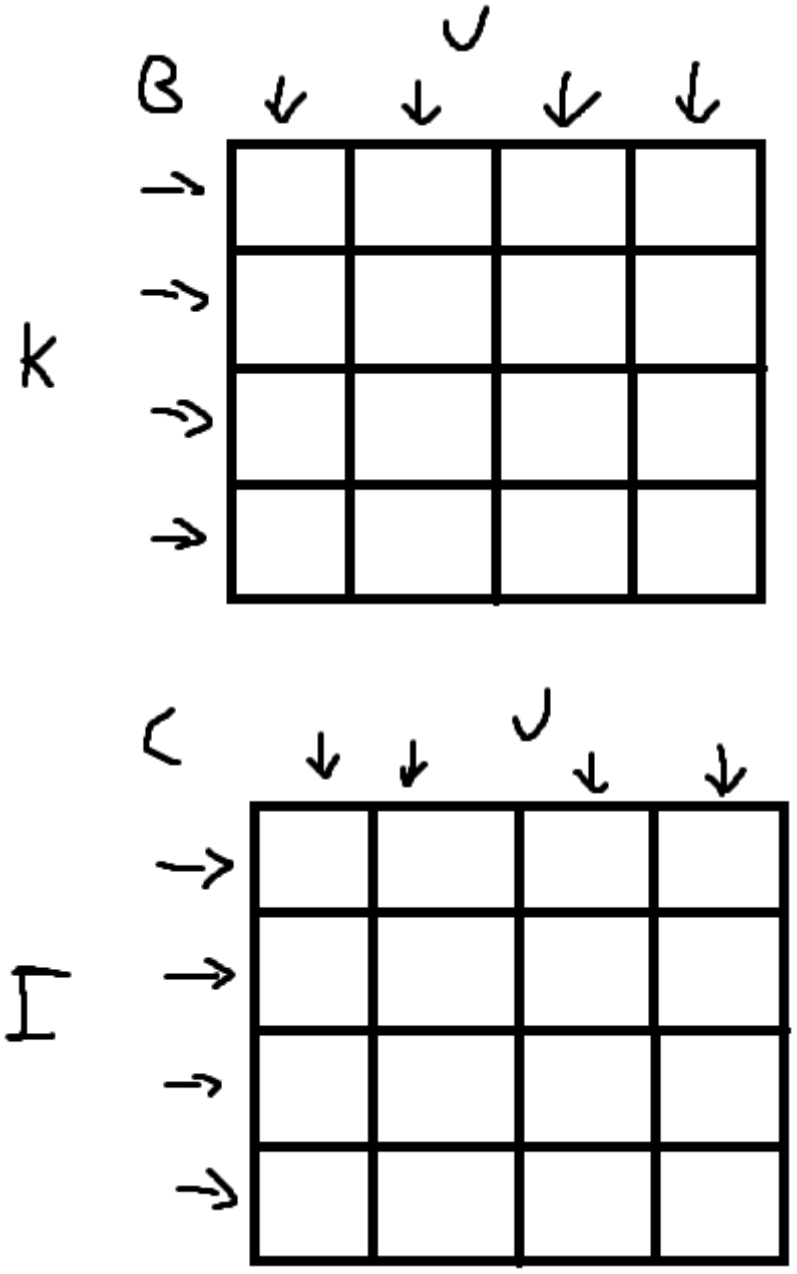
Antes de empezar a realizar los ejercicios vamos a entender que se está realizando para poder así implementarlo. Primero tenemos que tener en cuenta la fórmula que estamos realizando, es la siguiente:

```
C[i][j]= A[i][k]*B[k][j];
```

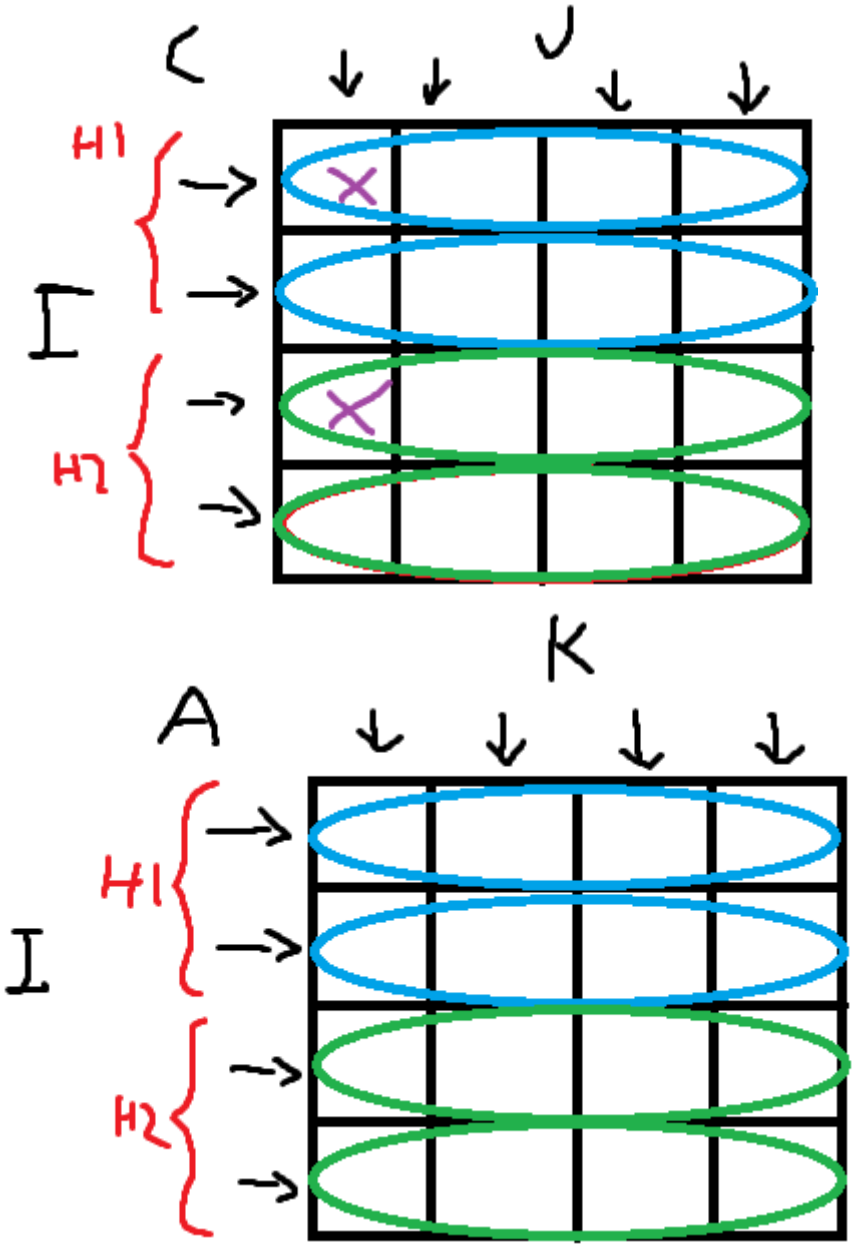
Tenemos que destacar que tanto esta fórmula es constante aunque cambiemos el orden de los bucles (i,j,k) o (i,k,j), el orden de los bucles afectará a la eficiencia como vimos en el ejercicio práctico uno.

Una vez que tenemos claro esto vamos a investigar cómo sería usar los hilos para el bucle i, k, j ya que este solo es necesario paralelizar el bucle i, y va a ser más sencillo entenderlo. Primeramente vamos a ver las matrices que tenemos y respecto a qué letra se referencian las columnas y las filas (pongo matrices 4x4 para simplificar):





Por tanto si paralelizamos sobre i, si tenemos una matriz de 2000x2000 y usamos dos hilos, cada hilo trabajaría con 1000 iteraciones en este caso serían 1000 filas. Para este dibujo serían dos y se vería reflejado en las matrices de la siguiente forma:



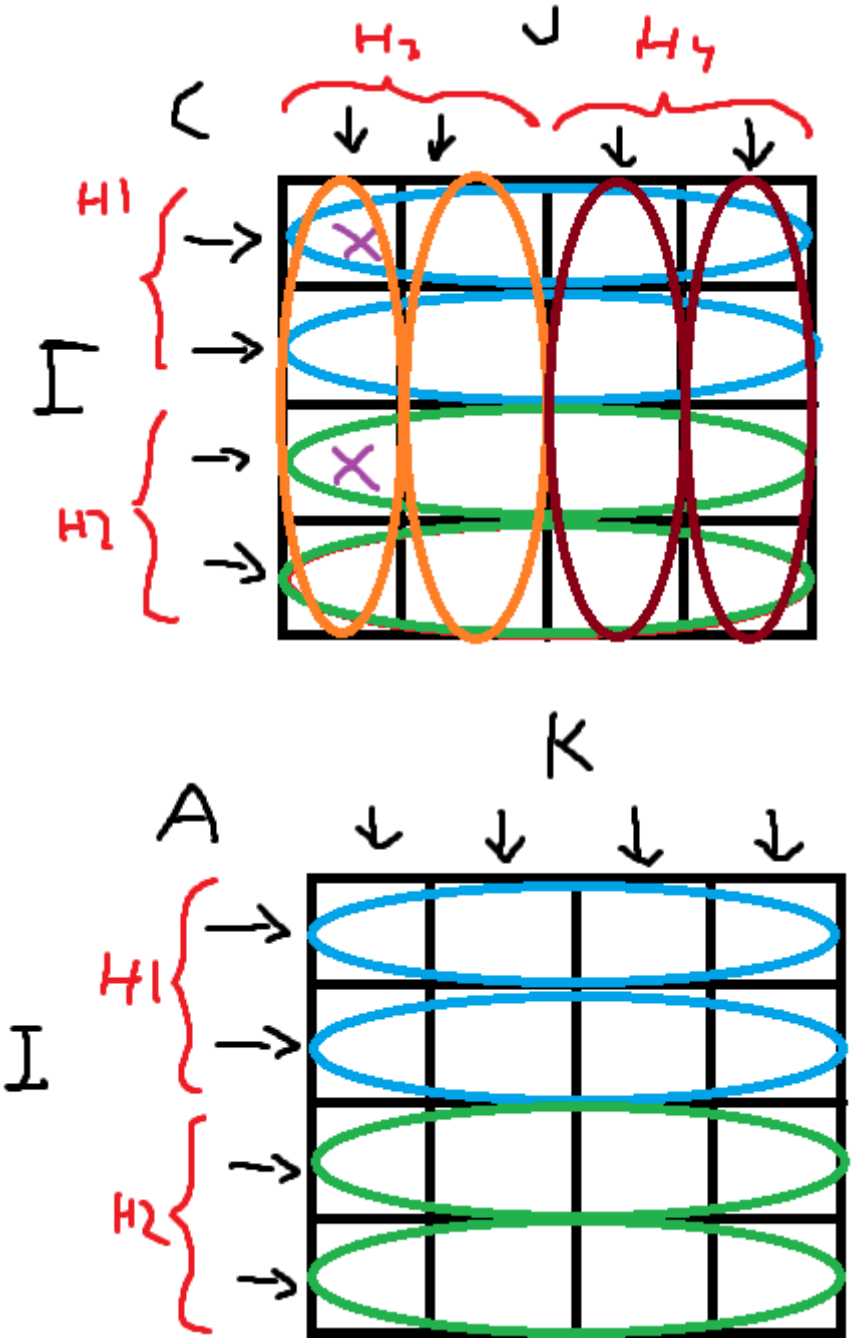
Respecto a la matriz B, no se paraleliza.
Lo que queremos entender con esto, es que la matriz I se dividirá en dos hilos. H1 se encargará de realizar las dos primeras filas mientras que H2 se encargará de realizar las dos últimas, esto se realizará de manera simultánea.
Respecto a la matriz A ocurre algo similar, entonces pongamos el ejemplo de rellenar la matriz I el elemento [0][0], ocurriría lo siguiente:
Si rellenamos este valor, el hilo H1 de la matriz A realizará las operaciones de la primera fila respecto a la fila correspondiente en la matriz B, esta no presenta hilos.

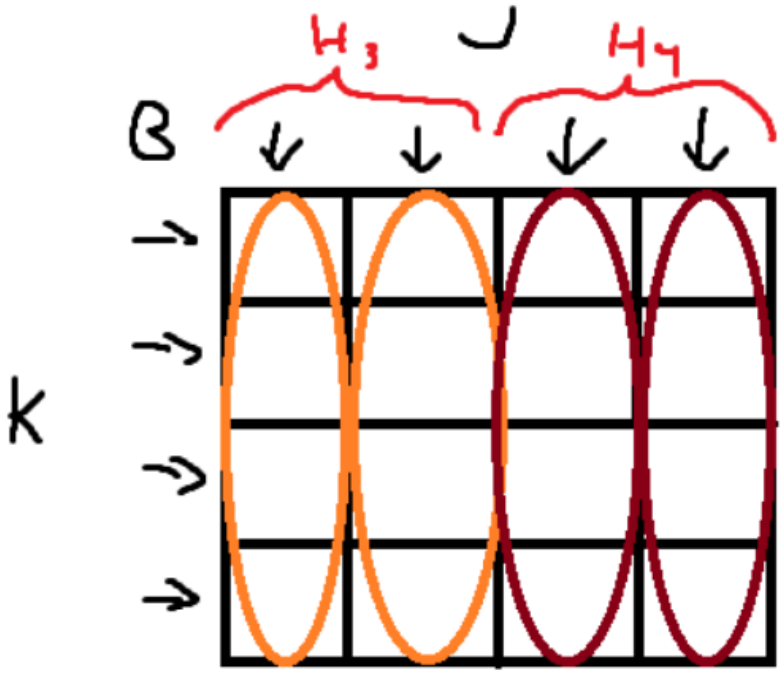
Con todo esto, permitimos que por ejemplo se pueda realizar la búsqueda del valor explicado anteriormente mientras que simultáneamente se realiza la búsqueda del valor I[2][0], ya que se realiza en un hilo diferente, en concreto en el hilo h2.

En este ejemplo se podrían utilizar hasta 4 hilos para el bucle i, pero hay que tener en cuenta que los hilo tendrán que esperar a que los datos de la matriz b estén libres ya que los hilos puede ser que intenten acceder al mismo dato.

En este caso creo que se podría llegar a paralelizar el bucle i y el k.

Ahora vamos a intentar comprender qué sucedería si intentamos paralelizar el bucle i,j,k.
En este caso pondremos las matrices directamente con los hilos, al paralelizar i y j:





Este caso parece más complicado, pero si lo entendemos es bastante sencillo, tanto la matriz A como la matriz B, se encuentran divididas a la mitad donde dos hilos se encargaran respectivamente, por tanto la matriz C, será una combinación de ambas viéndola, así como 4 partes. Para que se entienda, cuando llamemos al hilo que paralelizará todo esto (bucle i y j), sería de la siguiente forma (en este caso):
Parte1->i va desde 0 hasta 2 y j desde 0 hasta 2.
Parte2->i va desde 0 hasta 2 y j desde 2 hasta 4.
Parte3->i va desde 2 hasta 4 y j desde 0 hasta 2.
Parte4->i va desde 2 hasta 4 y j desde 2 hasta 4.

De, esta forma , se podrán realizar la multiplicación de la matriz de manera simultánea entre las cuatro partes, por ejemplo para calcular C[0][0] y C[3][3], C[0][0], usaría la primera fila del hilo 1 en a y la primera columna del hilo 3 en b, mientras que, en C[3][3] usaría la segunda fila del hilo 2 en a y la segunda columna del hilo 4 en b. De esta forma podrán trabajar de manera simultánea.

Ya una vez entendido el funcionamiento falta comprobar si realmente se aprecia una mejora en la práctica.

a) Elegir la paralelización sobre una o varias opciones de vectores de iteraciones, justificando la elección. (Una opción sería, por ejemplo, el vector de iteraciones (i, j, k) y paralelizar los bucles i y j. Otra opción sería el vector de iteraciones (i, k , j) y paralelizar el bucle i.)
Para este apartado se han realizado los dos tipos de paralizaciones explicados anteriormente, es decir, paralelización para i e j para iteraciones i,j,k y para paralelizar el

bucle i para iteraciones i,k,j. La implementación se puede observar de forma sencilla en el código, pero resumiendo, se basa en crear un hilo, en el caso de paralelizar i, donde se le pasa a ese hilo como debe recorrer el bucle i, de tal forma que si tenemos dos hilos y la matriz es de 4000x4000, entonces el bucle i se dividirá en dos partes. De esta forma cada hilo recorrerá 2000 filas.
Para i,j,k es similar pero hay que tener en cuenta que al paralelizar en 2 bucles necesitamos dividir entre 4, por tanto el número mínimo de hilos serán 4. De esta forma tendremos 4 hilos que se dedican de cada parte, respecto a i como j. El concepto con matrices de 4000x4000, sería el siguiente:
Parte1->i va desde 0 hasta 2000 y j desde 0 hasta 2000.
Parte2->i va desde 0 hasta 2000 y j desde 2000 hasta 4000.
Parte3->i va desde 2000 hasta 4000 y j desde 0 hasta 2000.
Parte4->i va desde 2000 hasta 4000 y j desde 2000 hasta 4000.

De esta forma en el mismo instante se pueden realizar varias operaciones.

Respecto a este punto, viendo el código y la explicación primera realizada se puede entender la implementación del paralelismo.

b) En función del vector de iteraciones elegido para la paralelización, ¿Es recomendable realizar algún cambio en las dimensiones de las matrices? Justificar la decisión.

Respecto a la teoría tenemos que decir:
Es muy recomendable realizar cambios a las dimensiones de las matrices a matrices de potencias de dos, es decir, en la práctica uno vimos que era bastante perjudicial pero en este caso nos es recomendable ya que como vimos, la caché funciona por bloques.
De tal forma que, si usamos la caché l1 de mi pc que vimos con anterioridad, los bloques pasan 16 datos, si mi matriz es de 4000x4000 y dividimos en dos hilos, tendríamos 2000 datos de tal forma que serían 125 bloques mientras que si son matrices de 4096x4096 y dividimos en dos hilos, con 2048 son 128 por tanto no hay problema.
Pero la l2 como vimos, es 2 x 1 MB 16-way, por tanto si analizamos una, son 1024/16=64, por tanto para cada way tenemos 64k, si el tamaño de línea que tenemos es de 64, entonces si dividimos nuestros 64k entre las 64 líneas, en total para cada línea tenemos 1024 bytes, si tenemos en cuenta que cada palabra son 4 bytes entonces tenemos por cada línea un total de 256 palabras. Por curiosidad, para cada way, caben 16.384 palabras.

A lo que queremos llegar con esto es que los bloques pasan 256 datos, si mi matriz es de 4000x4000 y dividimos en dos hilos, tendríamos 2000 datos de tal forma que serían 7,8125 bloques mientras que si son matrices de 4096x4096 y dividimos en dos hilos, con 2048 son 8.
Con esto, a lo que queremos llegar es que si usamos matrices que sean potencias de dos evitamos números decimales, vamos a explicar por qué es tan malo que un número de estos sea decimal.

Que un número sea decimal nos quiere decir que no se usa el bloque entero y esto es bastante problemático.

Esto se debe a que parte del bloque que tenemos en esta caché no se va a usar o peor, que se utilice para otra fila. Esto implica que si el procesador uno se encarga de x fila hasta la palabra 255 de un bloque y este lo coge de la caché dos y resulta que el procesador dos se encarga de x fila que empieza por el elemento 256 hasta el elemento 258, del primer bloque que vaya al procesador 1 se desperdicia un dato, esto no es tan problemático, pero la caché del procesador 2 tendrá 255 datos innecesarios y que aparte estos datos deben encontrarse sincronizados, aparte deberá de tener otro bloque para los datos restantes.
Este problema va a suceder siempre que nos encontremos una matriz que no sea potencia de dos, tras traer de la caché la primera fila ya que cogerá parte de la segunda que puede pertenecer a otro procesador.

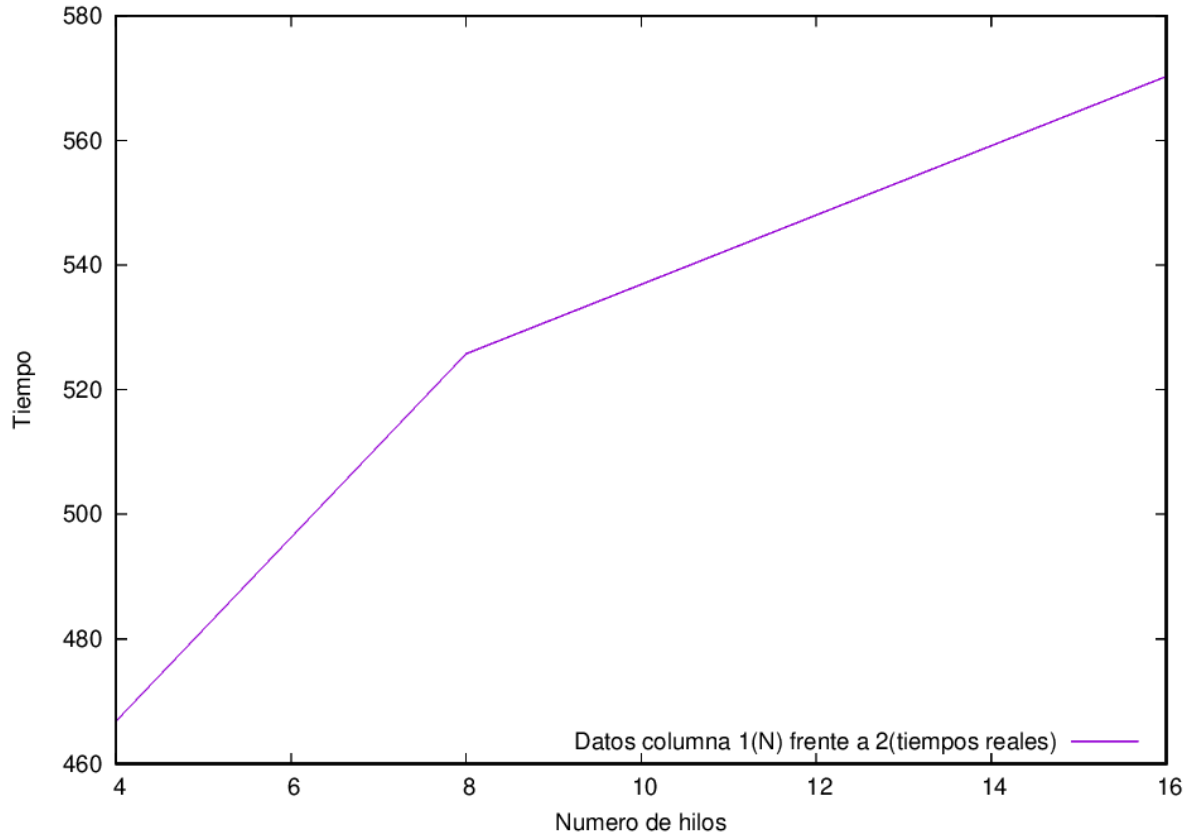
Una vez vista toda la teoría vamos a comprobar si realmente es cierto, pero esto se verá en el apartado de conclusiones ya que se verá de una forma más clara tras ver las gráficas con distintos números de hilos.

Preguntar mi caso de la I1 (donde no parece haber problemas con los bloques) y preguntar si esto solo ocurre en la transición entre procesadores, por ejemplo en la fila 2000.

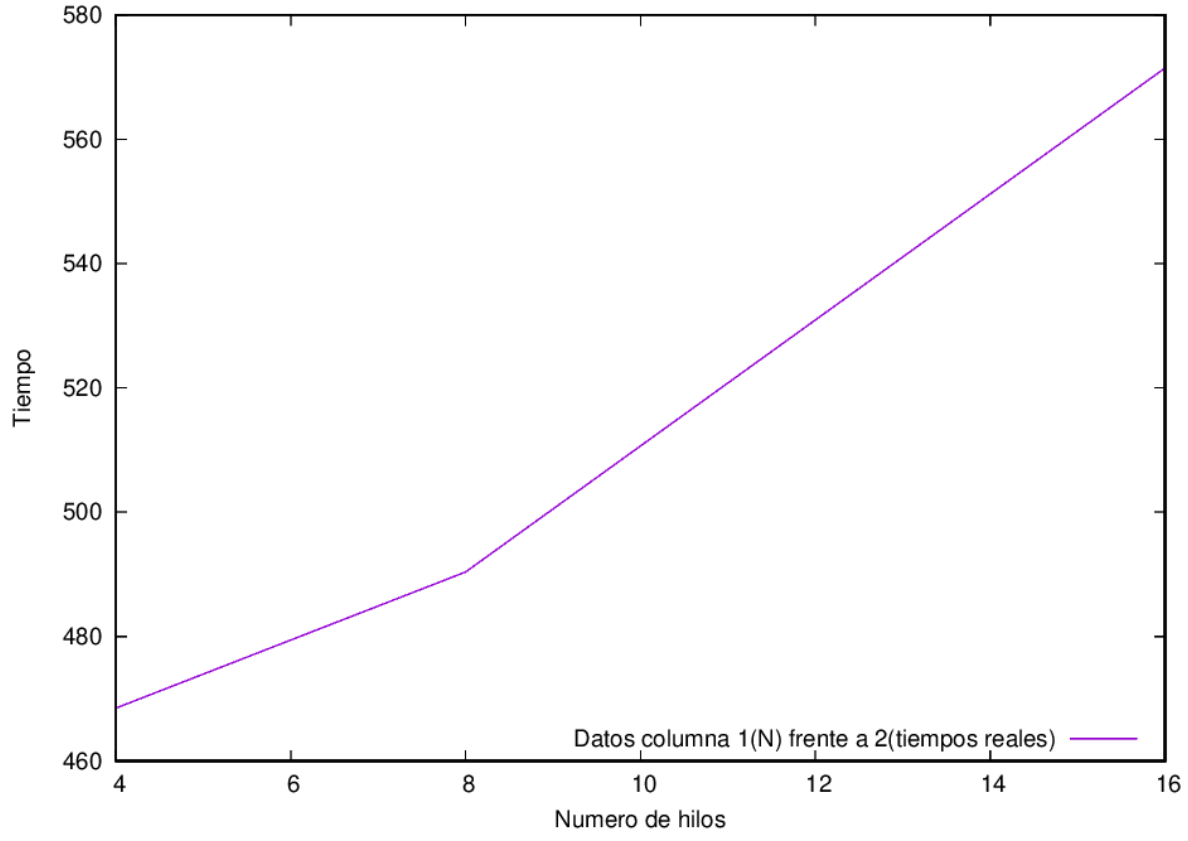
c) Para cada paralelización. Medir el tiempo de ejecución de la multiplicación contemplando la ejecución con distintos números de threads (2, 4, 8, 16, 32) y realizar curva de rendimientos o speed-up

En este apartado se mostrarán las gráficas para los distintos tipos de paralelización con los distintos tamaños de matriz, cabe destacar que como mi entorno de ejecución carece de 32 hilos no se han realizado pruebas con el mismo ya que son 4 núcleos con 4 hilos cada uno.
Para la paralización de ij no tiene sentido usar dos hilos ya que el mínimo serían 4 hilos, para usar mínimo dos hilos para i y dos hilos para j.

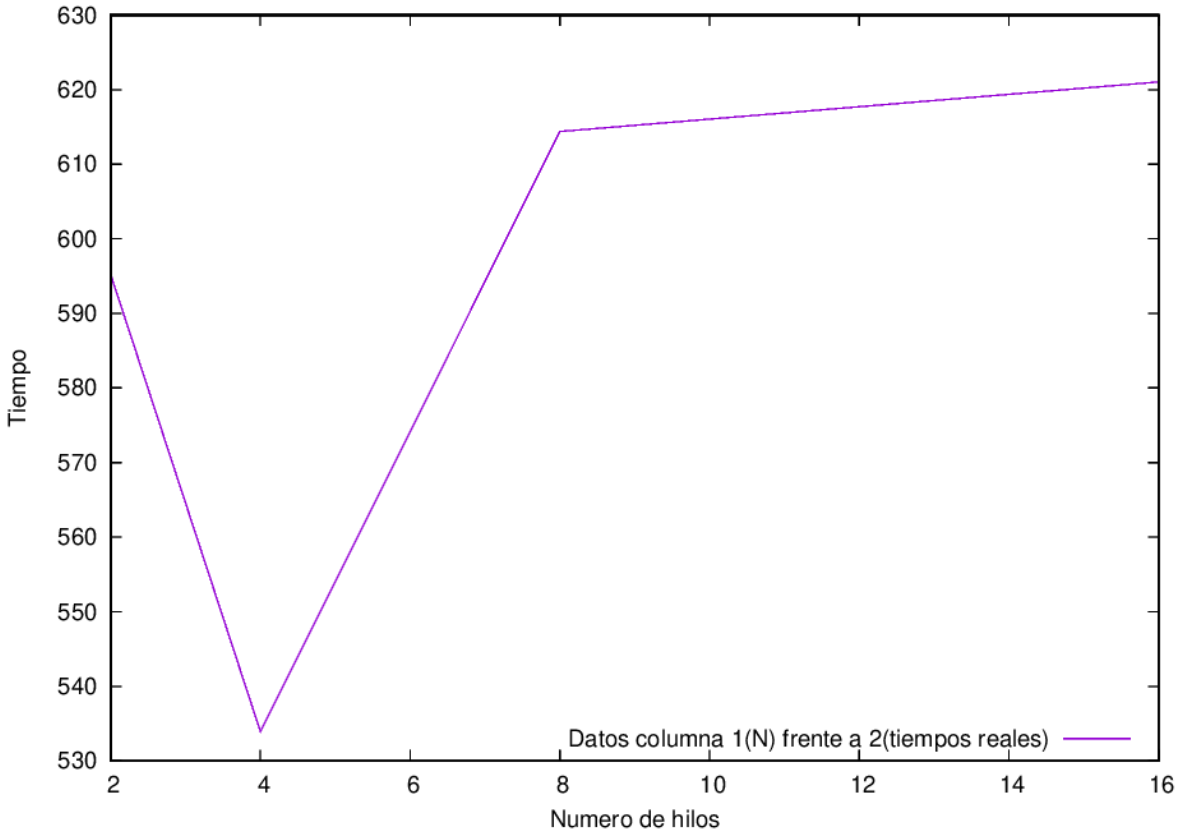
-Paralelización ijk con matrices de 4000:



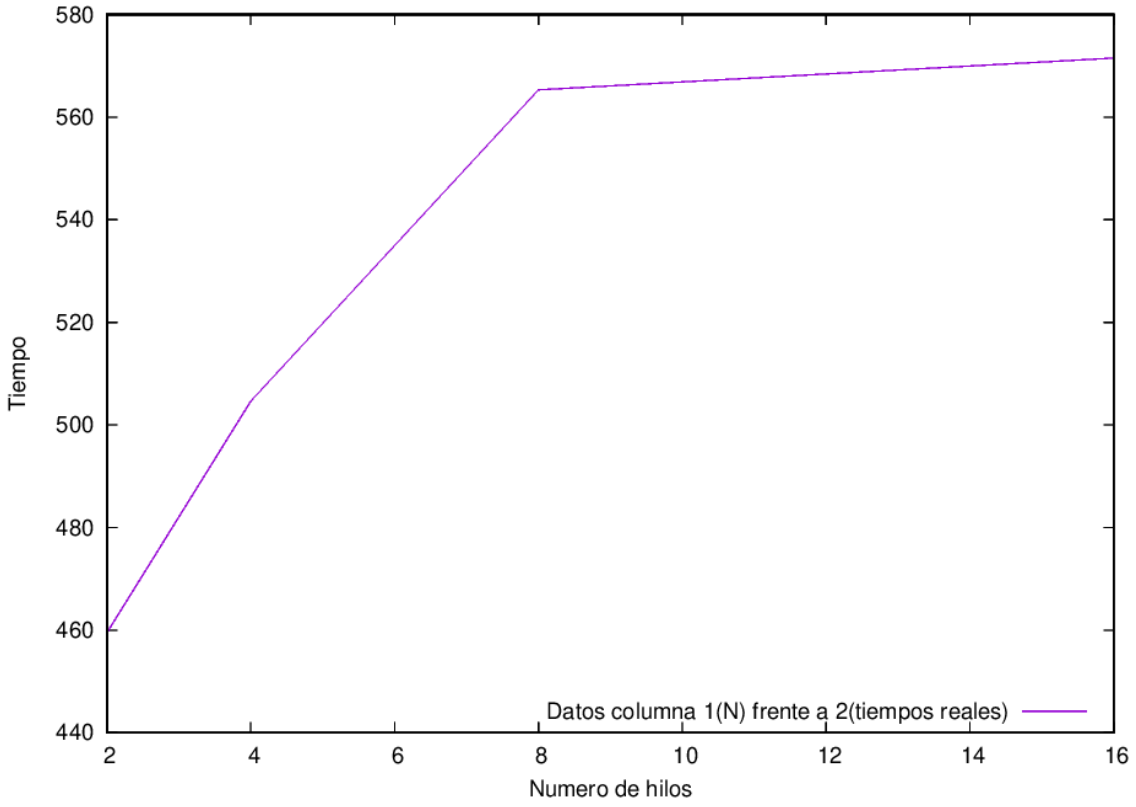
-Paralelización ijk con matrices de 4096:



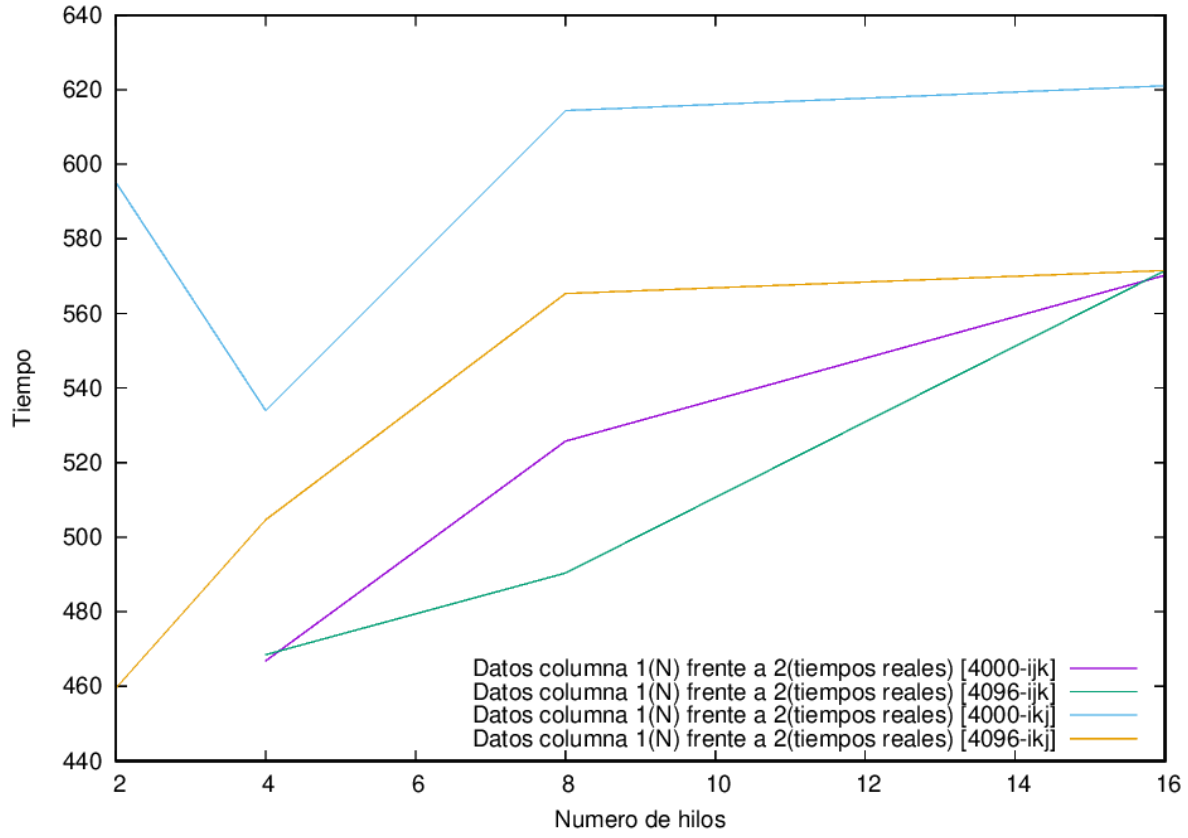
-Paralelización ikj con matrices de 4000:



-Paralelización ikj con matrices de 4096:



-Todas las gráficas en una:



d) Justificar los resultados obtenidos, detallando el entorno de ejecución.

Una vez vistas todas estas gráficas respecto a las pruebas en mi pc con las características comentadas en la práctica uno (compilador, l1, l2 y procesador con sus respectivos hilos), podemos llegar a las siguientes conclusiones:

Como vimos en el apartado 2, queda corroborado la explicación, ya que como podemos observar en las gráficas, el resultado obtenido al utilizar matrices de 4096x4096 siempre da resultados de tiempos menores, da igual el tipo de paralelización aplicada, que cuando aplicamos matrices de 4000x4000, esto tiene bastante sentido con la explicación dada en este apartado.

Respecto a los dos tipos de paralelización realizados, podemos observar que la paralelización realizada sobre las iteraciones ijk da mejores resultados que sobre la paralelización ikj. Esto se debe a que cuando paralelizamos sobre ikj solo paralelizamos sobre uno de los tres bucles mientras que con ijk paralelizamos sobre dos de los tres bucles, por tanto, permitimos realizar una mejor optimización partificando más los bucles y mejorando así la eficiencia.

Comparando el uso de hilos entre sí, suele ocurrir que a mayor funcionamiento peor rendimiento, esto se debe que internamente puede ocurrir que el hecho de crear muchos hilos y tener que gestionarlos por el sistema operativo puede reducir la eficiencia del programa, aun así hay algún caso que es mejor como por ejemplo ikj con 4000 que para 4 hilos es mejor que para dos hilos.

Por eso es necesario buscar un equilibrio entre cantidad de hilos con el programa para buscar el punto donde se equilibra la implementación de los hilos con la mejoría de tiempo.

Por último y como curiosidad sobre todo, se han realizado las pruebas con ijk con matrices de 4000x4000 y matrices de 4096x4096 para comparar tiempos y comprobar que si realmente vale la pena el tiempo invertido en paralelizar el programa.

Datos de bucle ijk en serie para matrices de 4000 valores:
957.320773
Datos de bucle ijk en serie para matrices de 4096 valores:
3224.076918

Como podemos observar, paralelizar para ijk es muy rentable ya que mejora el tiempo y tiene mucho sentido con lo explicado anteriormente, aparte podemos observar como vimos en la práctica 1 cuán perjudicial es usar matrices de potencias de dos para programas en serie.

También se realizaron las mismas pruebas para ikj obteniendo estos resultados:

Datos de bucle ikj en serie para matrices de 4000 valores:
407.826484
Datos de bucle ikj en serie para matrices de 4096 valores:
424.097162

En este caso no sale rentable paralelizar, de hecho da mejores tiempos en serie que en paralelo, lo cual no comprendo ni le encuentro la lógica, lo más lógico es que siguiera unos patrones similares al visto en ijk. Respecto a la práctica 1 sí que mantiene el concepto visto en la práctica uno y porque es peor con matrices con potencias de dos.

Como resumen general cabe destacar que la paralelización realiza una mejora respecto a un programa en serie (si aplicamos lo visto como usar matrices con potencias de dos sobretodo). Por consiguiente es útil, pero personalmente conlleva mucho tiempo y quebraderos de cabeza tener que paralelizar a mano el programa, a parte, debes tener muy en cuenta sobre qué arquitectura está trabajando para tener todo muy claro.
Lógicamente si buscamos la mayor eficiencia y no nos queda de otra, este es el mejor modelo ya que paralelizas todo a tu gusto y buscando la mayor eficiencia posible. Hasta que no pruebe otros métodos de paralelización no sabré concretamente cuan rentable es este método respecto al resto, me refiero a eficiencia y respecto a tiempo invertido en el programa.