Proyecto Final

Programación y Análisis de Algoritmos

Angel Adrián Castañeda Flores

13 de Diciembre de 2021

1 Objetivo

Los objetivos de este trabajo son los siguientes:

1. Programar en paralelo usando “pthreads”.
2. Programar en paralelo usando “cuda”.
3. Comparar tiempos de ejecución en diversos algoritmos en secuencial, “pthreads” y “cuda”.

2 Equipo Utilizado

Para este trabajo se hizo uso de una computadora portátil con las siguientes especificaciones:

1. *Procesador:* Ryzen 7 3750H.
2. *Tarjeta Gráfica:* Nvidia GTX 1650, 4 GB DDR5, 14 bloques, 896 CUDA, lo que representa 64 hilos por bloque.
3. *RAM:* 8 GB DDR4 a 2400 MHz.

3 Códigos Anexados

Los códigos entregados junto con este reporte generan matrices aleatorias entre [0-10], suman dos matrices, restan dos matrices, transpone una matriz y multiplica dos matrices y son los siguientes:

1. *”Proyecto_final_sec:”* código en c++ que realiza las operaciones mencionadas de manera secuencial. El código se entrega para utilizarse con un script “.sh”, las impresiones de los resultados vienen comentados.
2. *Proyecto_final_pthreads:* código en c++ que realiza las operaciones mencionadas en paralelo mediante el uso de pthreads. El código se entrega para utilizarse con un script “.sh”, las impresiones de los resultados vienen comentados y con un número de hilos igual a 8.
3. *Proyecto_final_cuda:* código en cuda que realiza las operaciones mencionadas en paralelo haciendo uso de una tarjeta gráfica Nvidia. El código se entrega con las impresiones de los resultados comentados.

4. *Pruebas_final_sec:* código en bash/shell que automatiza las pruebas de "Proyecto_final_sec" para diversos tamaños de matrices.

5. *Pruebas_final_pthreads:* código en bash/shell que automatiza las pruebas de "Proyecto_final_pthreads" para diversos tamaños de matrices, el número de hilos se cambia manualmente.

4 Algoritmos

Los algoritmos programados fueron operaciones entre matrices. Las operaciones programadas son suma, resta, multiplicación y la transpuesta de una matriz.
Los tres códigos cuentan con un orden similar para facilitar la comparación de rendimiento. Por ejemplo, cada uno toma dimensiones para las matrices A y B, para posteriormente llenarlas con valores aleatorios entre (0,10].
Una tercera matriz C es generada al momento de que se realizan operaciones entre matrices y se le asignan las dimensiones correspondientes una vez que se verifica que la operación correspondiente es ejecutable. Después de imprimir el resultado la matriz C es "eliminada".
Las formulas utilizadas para cada operación son las siguientes:
Suma de A con B:

$$C_{ij} = A_{ij} + B_{ij} \tag{1}$$

Resta de A con B:

$$C_{ij} = A_{ij} - B_{ij} \tag{2}$$

Transpuesta de A:

$$C_{ij} = A_{ji} \tag{3}$$

Multiplicación A*B:

$$C_{ij} = (A_{ik} * B_{kj}) + C_{ij} \tag{4}$$

Más adelante en este reporte comentaré particularidades acerca de los códigos en paralelo. Respecto al código en secuencial lo único que puedo mencionar es que realiza las operaciones requeridas, no utiliza funciones. Todas las operaciones se encuentran del programa principal.

4.1 Memoria Dinámica y Vectorización de Matrices

La memoria dinámica es el eje de estos códigos para que estuvieran en condiciones similares. De manera contraria a como yo he declarado previamente en otras tareas una matriz de dimensiones "dinámicas", la cual consistía en un puntero doble, en este trabajo se optó por utilizar un puntero con una longitud de: $columnas * filas$. En otras palabras, en vez de trabajar con una matriz, estoy trabajando con un vector y por medio de un artificio puedo acceder a sus elementos como si fuera una matriz.
Para acceder a cada elemento de la matriz, independientemente de que fuera declarada como vector se hace uso de la siguiente ecuación:

$$elemento_{ij} = i * NColumnas + j \tag{5}$$

Donde i va de 0 al número de filas y j de 0 al número de columnas.
Esta manera de expresar las matrices, en lo personal, facilita la asignación y liberación de memoria.

5 Pthreads

Dentro del código de "pthreads", se hace comentario de que se paralelizara por filas y se comienza la generación de limites, por ejemplo de las líneas 141-150.
Dentro de mi concepción de paralelizar, cada hilo debe ejecutar el mismo número de operaciones correspondientes a $nfilas/nhilos$ número de filas con la finalidad de obtener un resultado de manera más rápida.
En el caso que hubiera menos filas que hilos, solo un hilo trabaja. En caso que el número de filas fuera múltiplo del número de hilos, cada hilo realiza la misma cantidad de operaciones. Finalmente, si el número de filas no es múltiplo del numero de filas, un hilo trabaja una operación más o una operación menos dependiendo el caso.
La repartición de filas o tareas entre cada hilo se realiza con la siguiente formula:

$$axuint = redondear(filasA/n_threads) \tag{6}$$

Calculado "auxint", se genera un vector de límites con dimensión $nhilos + 1$, este vector comienza en 0 y a cada elemento posterior es calculado como $elementoanterior + auxint$. Este vector es el que ayuda a definir las filas en que cada hilo va a trabajar.

Finalmente, dentro de este código tengo dos funciones comentadas (CrearMA y CrearMB), algunas líneas de código comentadas y las variables "contadorA" y "contadorB" comentadas. Esto debido a que como ejercicio extra había paralelizado la generación de números aleatorios para las matrices A y B.

6 Cuda

A diferencia de "pthreads", "cuda" realiza una operación por hilo. En otras palabras, realiza operaciones elemento a elemento. Lo que implica esto es que en las funciones de "cuda" se eliminen dos ciclos "for".
Para comenzar a usar cuda se debe reservar memoria en la tarjeta gráfica y copiar los valores de las variables guardadas en el CPU al GPU. Para reservar memoria en la memoria gráfica se hace uso del siguiente comando:

$$cudaMalloc((void)\&cudaA, filasAcolumnasAsizeof(int)); \quad (7)$$

En el comando de arriba, se reserva memoria en el GPU con las dimensiones de una matriz. Para copiar los valores del CPU al GPU se hace uso de siguiente comando:

$cudaMemcpy(cudaA, matrizA, filasA * columnasA * sizeof(int),$
$cudaMemcpyHostToDevice);$

Con el comando anterior, se copia la información de matriz A (dentro del CPU) a una variable dentro del GPU (cudaA).
La parte más importante fue la definición de los números de hilos por bloque y el número de bloques a utilizar. Partiendo de las especificaciones de la tarjeta gráfica se define un total de 64 hilos por bloque, por lo que se uso un arreglo de $8x8$.

$$dim3threadsPerBlock(8, 8); \quad (8)$$

El número de bloques a utilizar se define como:
$dim3numBlocks(ceil(float(filasC)$
$float(threadsPerBlock.x)), ceil(float(columnasC)/float(threadsPerBlock.y)))$

Para ejecutar una función directamente sobre el GPU se llama desde main y tiene la siguiente estructura
$NombreFuncion <<< NumerodeBloques, HilosporBloque >>>$
$(Argumentosdelafuncion)$.
La función en "cuda" se declara fuera del main (como cualquier otra función)

y tiene la siguiente estructura:
$\_\_global\_\_void(variables)\{$
$cuerpodelafuncion\}$
Posterior a llamar una función es buena práctica usar el comando $cudaDeviceSynchronize()$ para esperar a que el GPU termine de ejecutar todas sus operaciones.
Para copiar información del GPU al CPU se hace uso del comando:
$cudaMemcpy(matrizC, cudaC, filasCcolumnasCsizeof(int),$
$cudaMemcpyDeviceToHost)$;, en este comando se copia la información de la variable cudaC (dentro del GPU) a la variable MatrizC (dentro del CPU).
Finalmente, se libera la memoria asignada en el GPU con:
$Free(NombredeVariable)$.

7 Pruebas

Las pruebas de cada código se dividió en dos etapas, una para la verificación de resultados y otra para ver tiempos de ejecución.

- *Verificación de resultados*, se toman matrices de dimensiones pequeñas para verificar que cada operación sea ejecutada de manera correcta. Las dimensiones son las siguientes: MatrizA 3x2,2x3 y 3x3, cada dimensión con las siguientes dimensiones de la matrizB 2x3, 3x2 y 3x3. Para ver algunas pruebas, ver punto 10.2.

- *Tiempos de Ejecución*, se corrió cada código para diversos tamaños de matrices cuadradas. En este caso ya no se imprimen los resultados, solo los tiempos que se tarda en realizar cada operación. Los tiempos se cuentan desde la declaración de las dimensiones de la matriz resultado y asignación de memoria y se deja de contar en el momento que se libera su memoria.

7.1 Igualdad de Condiciones en las Pruebas de Tiempo

Para que los códigos fueran los más similares posibles y ver solamente el efecto de la paralelización en "pthreads" o en "cuda" se tomaron las siguientes consideraciones:

- El conteo comienza desde que se definen las dimensiones de la matriz C.

- El conteo se detiene cuando de libera la memoria asignada a la matriz C.
- Los números aleatorios dentro de la matriz A y B se encuentran en el intervalo [0,10].
- En los tres programas se utilizó la misma asignación de memoria y acceso a elementos para cada matriz como mencionado en el apartado Memoria Dinámica.

7.2 Automatización de Pruebas

Los códigos de las operaciones en secuencial y con "pthreads" se pudieron automatizar para los pruebas de tiempo mediante un "script" en "bash". Dichos "scripts" ejecutaban las pruebas para matrices cuadradas de tamaños 100, 500, 1000, 1500, 2000, 2500, 3000, 4000 y 5000. El "script" para el código en secuencial se ejecutó una vez que es el número de pruebas por dimensión que se planteo en el apartado de pruebas. El "script" para "pthreads" se ejecutó una vez por cada número de hilos a probar. El "script" no asigna el número de hilos por su cuenta. Respecto al código en "cuda" no pude automatizar sus pruebas.

8 Resultados de Tiempos de Ejecución

Esta sección solo muestra tablas con los tiempos de ejecución de los algoritmos y gráficas junto a una breve interpretación. Para ver las gráficas, ir al Punto 10.1.

Tiempos Suma en segundos					
n	Secuencial	Pthreads8	Pthreads4	Pthreads2	Cuda
100	0	0	0	0	0.000187168
500	0	0	0	0	0.000972736
1000	0	0	0	0.015625	0.0033561
1500	0.015625	0.00390625	0.015625	0.015625	0.00514605
2000	0.03125	0.015625	0.015625	0.03125	0.00818445
2500	0.03125	0.00976562	0.03125	0.03125	0.0236028
3000	0.046875	0.03125	0.03125	0.0390625	0.0216204
4000	0.125	0.0410156	0.0507812	0.0625	0.0503467
5000	0.171875	0.0761719	0.0664062	0.101562	0.0539919

Tiempos Resta en segundos					
n	Secuencial	Pthreads8	Pthreads4	Pthreads2	Cuda
100	0	0	0	0	0.000202592
500	0	0	0	0	0.00126518
1000	0	0.0177188	0	0	0.00511904
1500	0.03125	0.00585938	0	0.015625	0.00925286
2000	0.03125	0.0136719	0.015625	0.015625	0.0126007
2500	0.03125	0.015625	0.0195312	0.03125	0.0231773
3000	0.0625	0.03125	0.0351562	0.03125	0.029644
4000	0.125	0.0390625	0.0507812	0.0625	0.0630395
5000	0.1875	0.103125	0.09375	0.117188	0.0796273

Tiempos Transpuesta en segundos					
n	Secuencial	Pthreads8	Pthreads4	Pthreads2	Cuda
100	0	0	0	0	0.000161792
500	0	0	0	0.015625	0.0011192
1000	0	0	0.015625	0	0.00536646
1500	0	0	0.015625	0	0.00828112
2000	0.015625	0	0.0117188	0.0078125	0.0128217
2500	0.046875	0.00195312	0.00390625	0.015625	0.0202138
3000	0.078125	0.0273438	0.0390625	0.0703125	0.0265689
4000	0.125	0.0820312	0.0625	0.078125	0.0581181
5000	0.21875	0.115234	0.101562	0.132812	0.0703201

Tiempos Multiplicación en segundos					
n	Secuencial	Pthreads8	Pthreads4	Pthreads2	Cuda
100	0	0	0	0	0.000252224
500	0.359375	0.07226556	0.09375	0.15625	0.0095472
1000	3.125	0.630859	0.855469	1.60938	0.0663965
1500	13.7031	2.86523	3.60547	6.54688	0.221002
2000	32.5469	7.3125	9.59375	17.1172	0.515678
2500	84.9219	16.1973	23.9609	41.25	0.950295
3000	131.359	30.7695	38.9336	69.4766	1.56592
4000	311.484	82.0957	100.055	177.133	3.60689
5000	613.359	207.691	212.445	344.406	7.22856

9 Conclusiones

- Para el caso de la Suma, Resta y Transpuesta no hay diferencia apreciable entre las formas de programar hasta que se toma una matriz de tamaño *3000x3000*.
- En el caso de la multiplicación, se aprecia un cambio notorio entre un código secuencial y uno paralelizado.
- Hay una diferencia apreciable entre "pthreads" con 2 hilos a "pthreads" con 2 y 4 hilos, sobretodo en la multiplicación.
- A pesar que si hay una diferencia entre usar 4 y 8 hilos en "pthreads", esta no parece ser tan grande.
- el punto anterior puede deberse al procesador utilizado para las pruebas.
- En general, la paralelización en "cuda" es la más rápida.
- Una tarjeta gráfica tiene una frecuencia base mucho menor a un CPU, es menos potente. Lo que hace que la paralelización en *cuda* sea superior a la paralelización en CPU es la enorme diferencia entre el número de hilos que se pueden ocupar.
- Una mala definición del mallado para paralelizar en *cuda* puede llevar a errores y el programa aborta la ejecución del código.
- Expresar una matriz como vector facilita mucho el manejo de memoria y el paso de variables a funciones.

10 Anexos

10.1 Gráficas de Tiempo de Ejecución

En esta sección se muestran las gráficas de los tiempos de la sección 8.

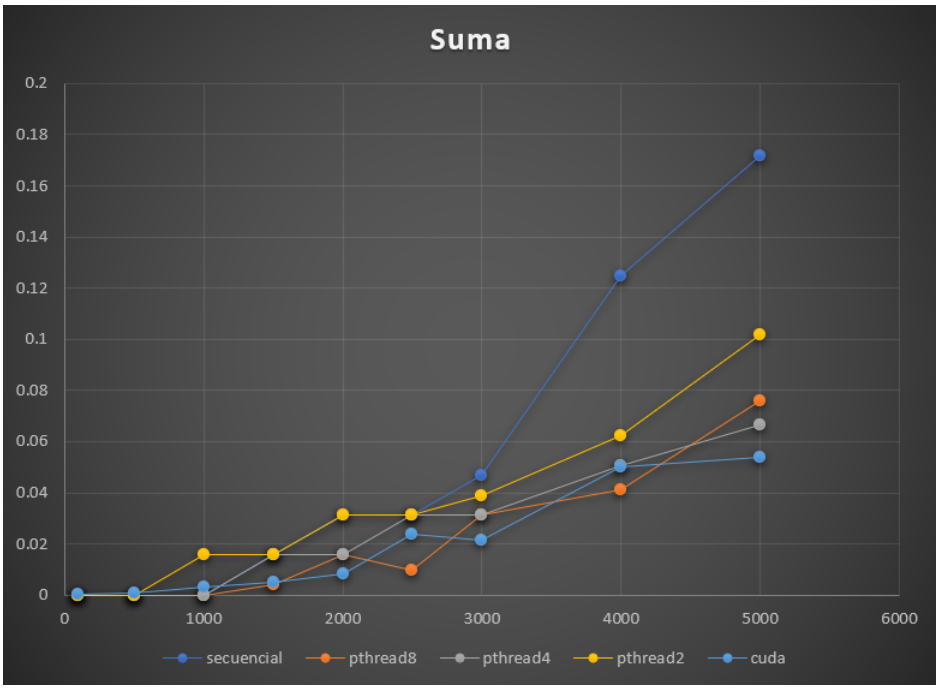

Figure 1: Gráfica de tiempos para la suma.

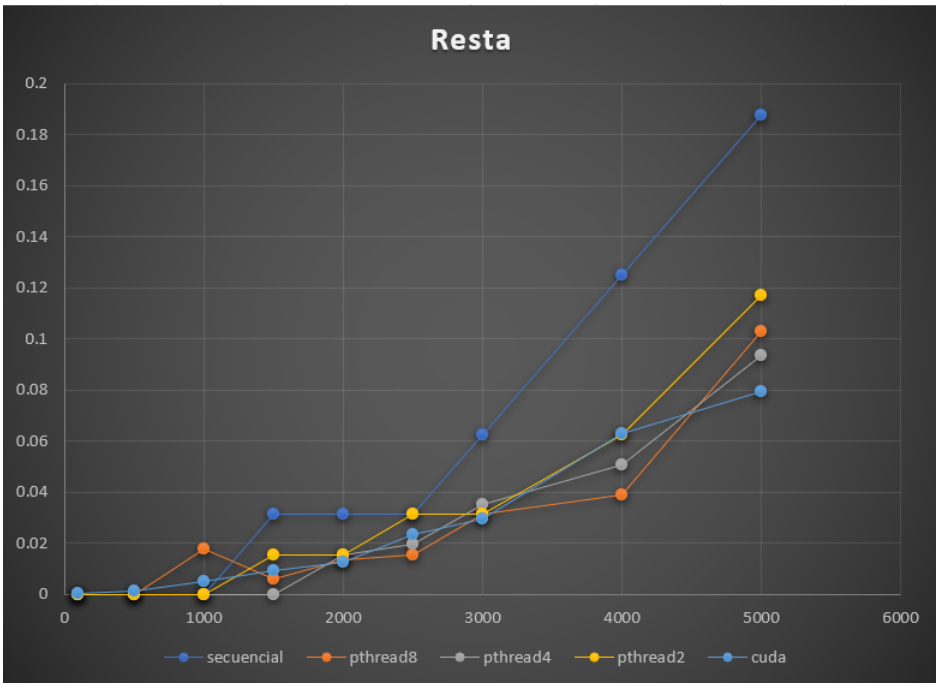

Figure 2: Gráfica de tiempos para la resta.

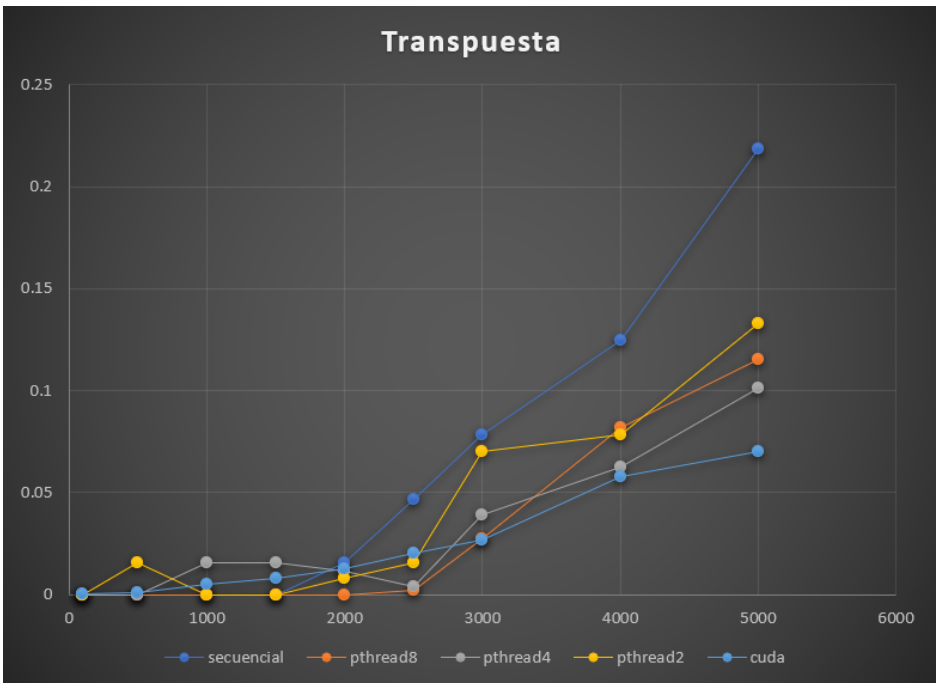

Figure 3: Gráfica de tiempos para la transpuesta.

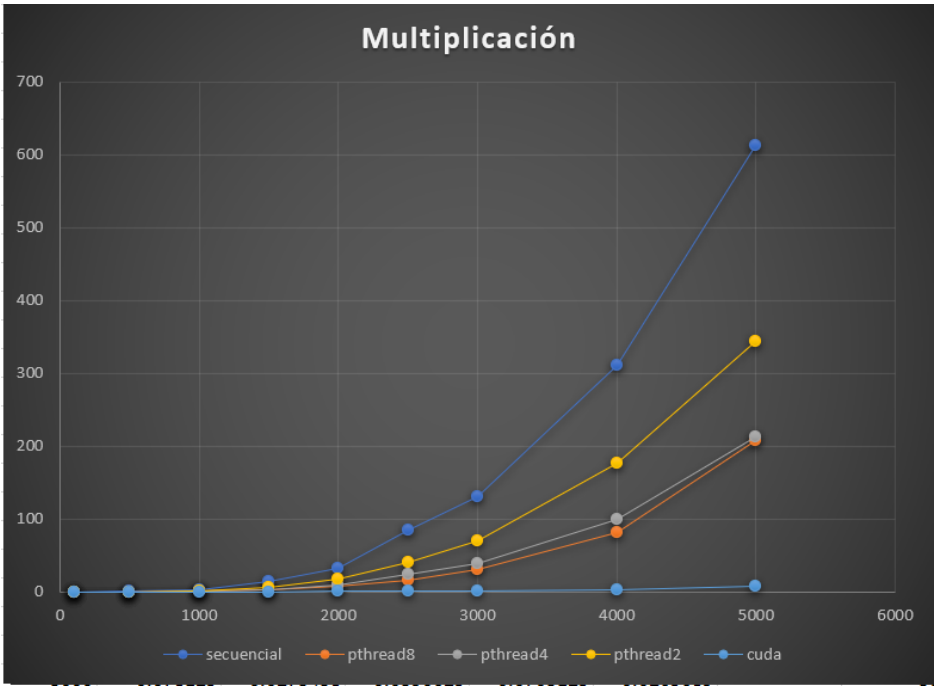

Figure 4: Gráfica de tiempos para la multiplicación.

10.2 Pruebas de Verificación de Resultados

Se anexan capturas de que cada código ejecuta debidamente su operación.

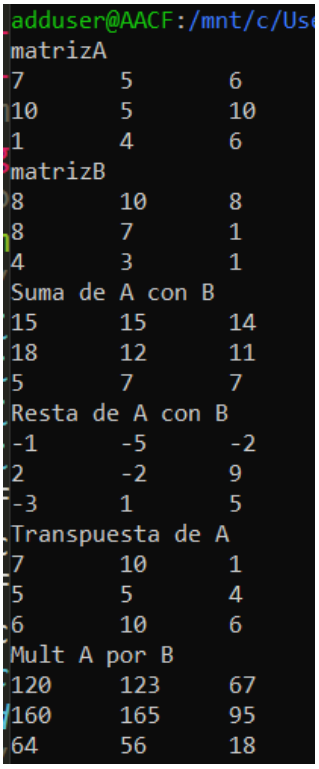

Figure 5: Resultados Secuencial matriz A de 3x2 y matriz B 2x3.

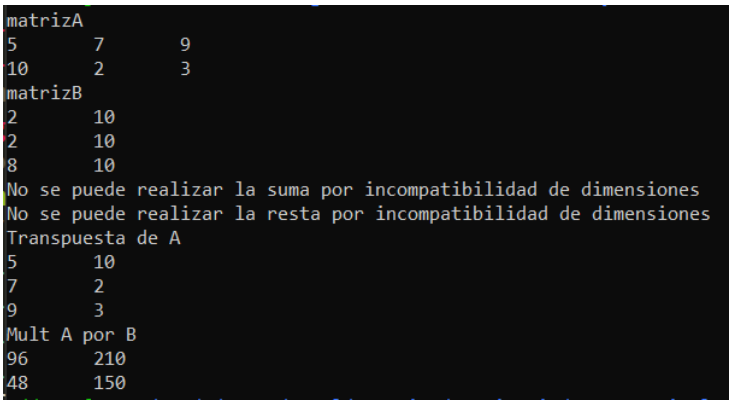

Figure 6: Resultados Secuencial matriz A de 2x3 y matriz B 3x2.

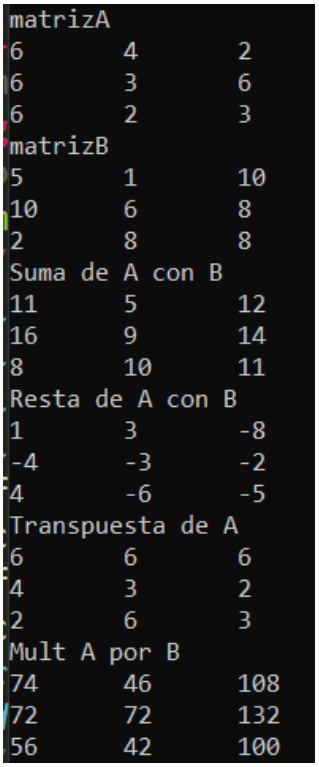

Figure 7: Resultados Secuencial matriz A de 3x3 y matriz B 3x3.

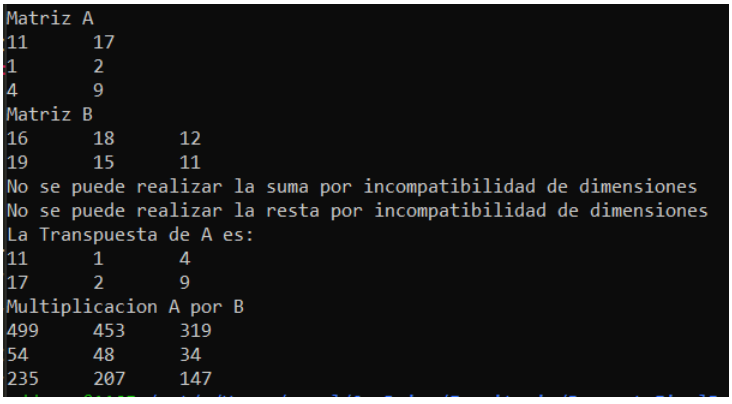

Figure 8: Resultados pthread matriz A de 3x2 y matriz B 2x3.

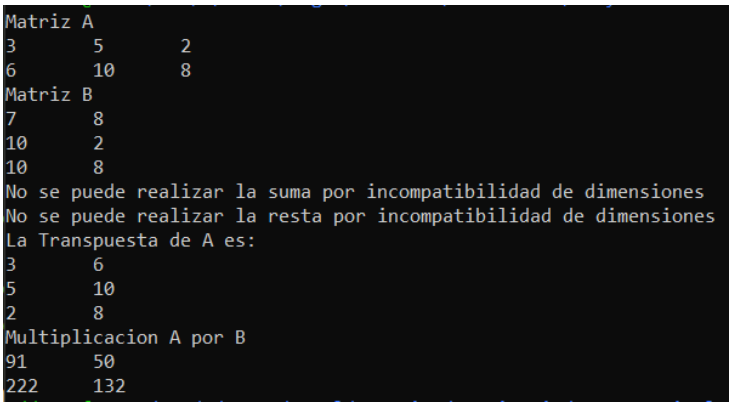

Figure 9: Resultados pthread matriz A de 2x3 y matriz B 3x2.

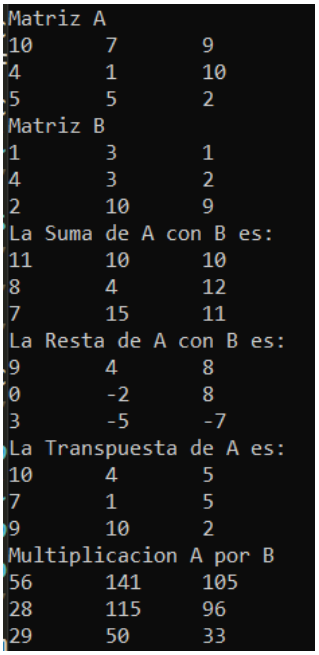

Figure 10: Resultados pthread matriz A de 3x3 y matriz B 3x3.

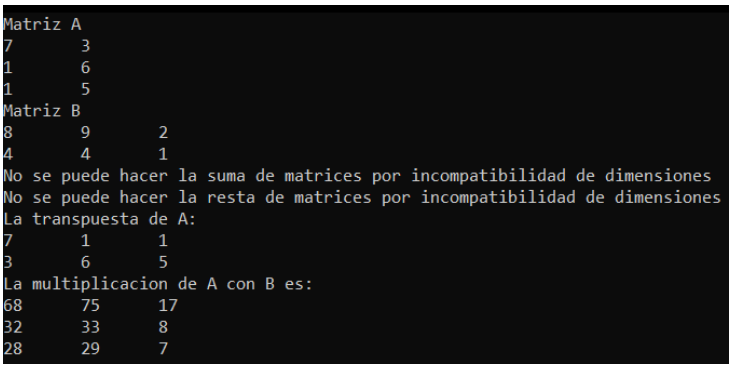

Figure 11: Resultados cuda matriz A de 3x2 y matriz B 2x3.

```
Matriz A
1       5       6
6       2       4
Matriz B
9       5
1       2
9       3
No se puede hacer la suma de matrices por incompatibilidad de dimensiones
No se puede hacer la resta de matrices por incompatibilidad de dimensiones
La transpuesta de A:
1       6
5       2
6       4
La multiplicacion de A con B es:
68      33
92      46
```

Figure 12: Resultados cuda matriz A de 2x3 y matriz B 3x2.

```
Matriz A
2       8       9
3       4       1
1       5       1
Matriz B
10      4       9
6       5       3
10      3       8
La suma de A con B es:
12      12      18
9       9       4
11      8       9
La resta de A con B es:
-8      4       0
-3      -1      -2
-9      2       -7
La transpuesta de A:
2       3       1
8       4       5
9       1       1
La multiplicacion de A con B es:
158     75      114
64      35      47
50      32      32
```

Figure 13: Resultados cuda matriz A de 3x3 y matriz B 3x3.