Luciano Pulido
DNI:40137604

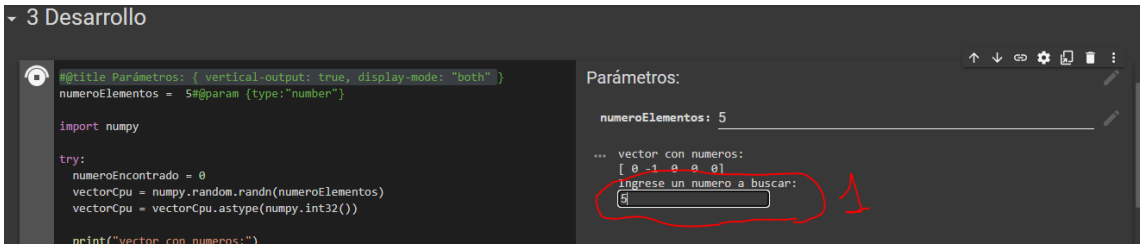
Explicación general ejercicio 1:

En este ejercicio se desarrolló el algoritmo de búsqueda secuencial tanto usando solamente la CPU de manera secuencial como usando la GPU aplicando paralelismo. En este ejercicio lo que se hace es ingresar la cantidad de elementos que deseamos que tenga un vector y luego ingresamos un número que deseamos que se busque dentro del vector y se nos informe si se encontró o no. El objetivo buscado con este ejercicio es comparar los tiempos de ejecución de la manera secuencial contra la manera que utiliza paralelismo para ver cual resulta ser más eficiente.

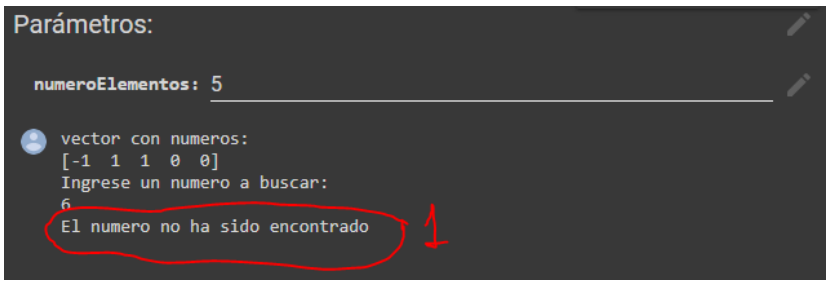
Manual de uso para la versión CPU (secuencial):

Paso1: Dentro del cuaderno, debemos dirigirnos a la sección “Desarrollo” y luego ingresar el número de elementos que deseamos que tenga el vector donde se va a buscar un número, como se indica donde dice “1 (en rojo)” señalado en la imagen, luego debemos hacer click en el botón de “play” donde dice “2 (en rojo)” de la imagen.

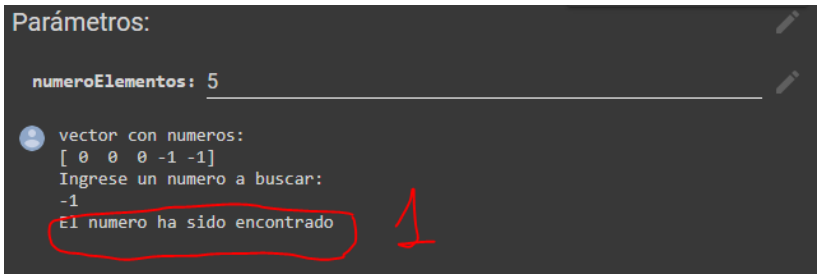
Paso2: Luego se nos solicitara por pantalla que ingresemos un numero a buscar en el vector creado previamente con números enteros al azar, ingresamos un numero como se indica donde dice “1 (en rojo)” y después presionamos enter



Paso3a: Si el número ingresado por teclado no fue encontrado en el vector se muestra un mensaje como se indica donde dice “1 (en rojo)



Paso3b: Si el número ingresado por teclado fue encontrado en el vector se muestra un mensaje como se indica donde dice “1 (en rojo)

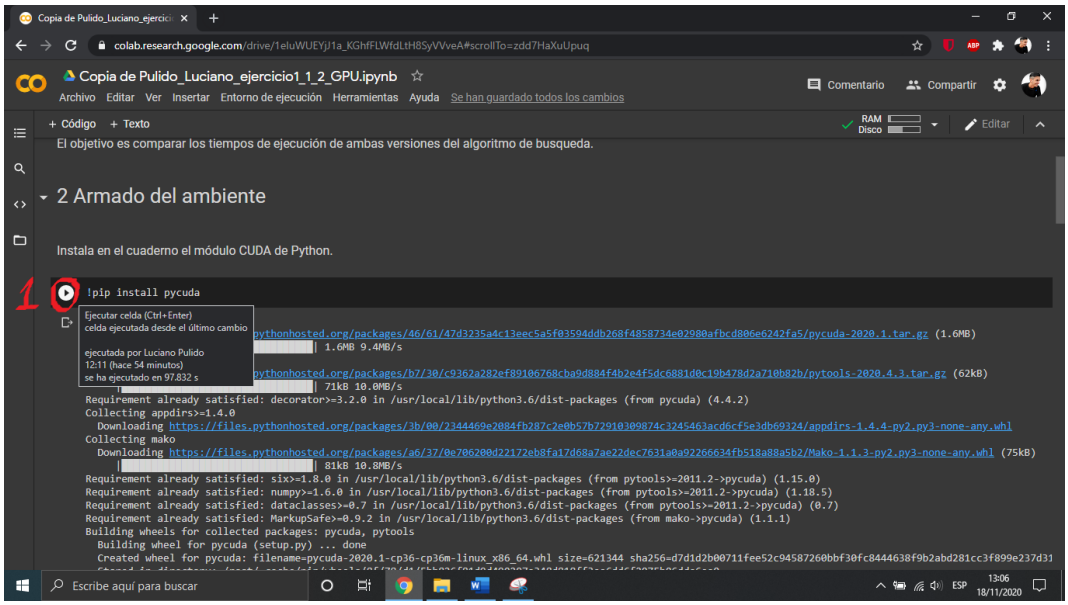


link versión secuencial: https://github.com/lucianopulido/EA2-Luciano-Pulido/blob/master/HPC/Pulido_Luciano_ejercicio1_1_2_CPU.ipynb

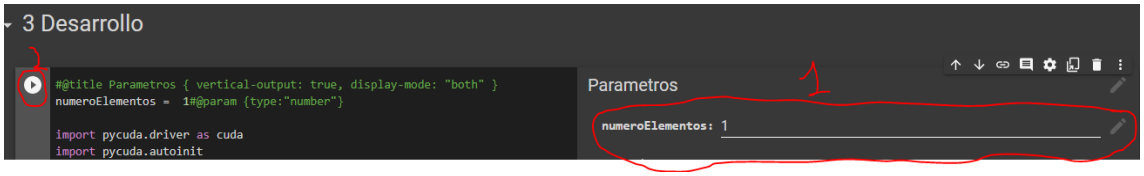
Manual de uso para la versión GPU (paralelismo):

Aclaración: esta versión debe ser ejecutada en el entorno de GPU.

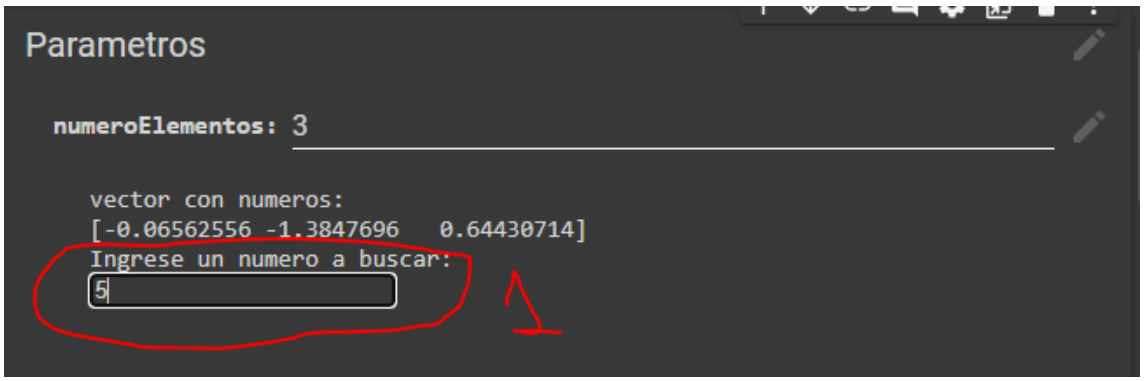
Paso1: : Dentro del cuaderno, debemos dirigirnos a la sección “Armado del ambiente” y luego debemos hacer click en el botón de “play” donde dice “1 (en rojo)” de la imagen.



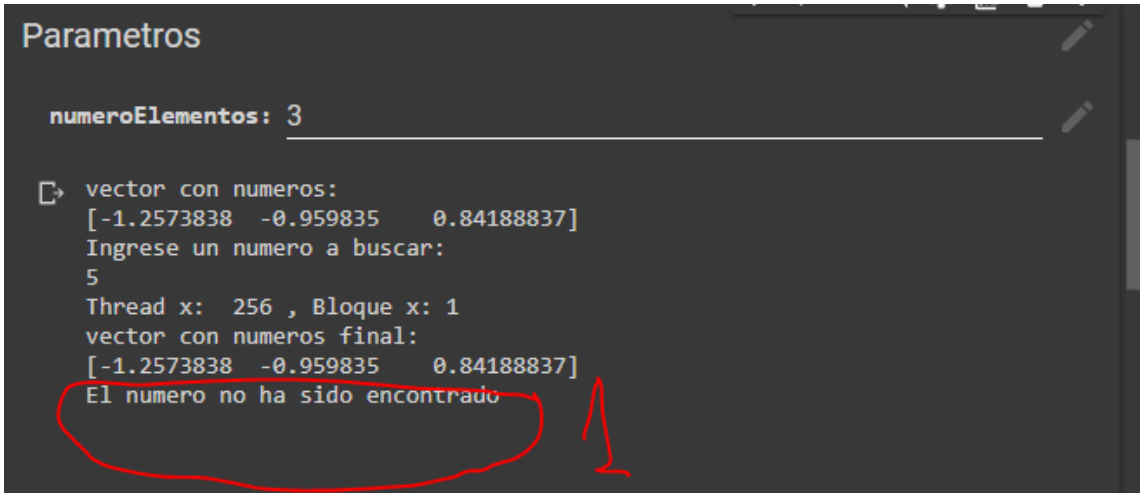
Paso2: Dentro del cuaderno, debemos dirigirnos a la sección “Desarrollo” y luego ingresar el número de elementos que deseamos que tenga el vector donde se va a buscar un número, como se indica donde dice “1 (en rojo)” señalado en la imagen, luego debemos hacer click en el botón de “play” donde dice “2 (en rojo)” en la imagen



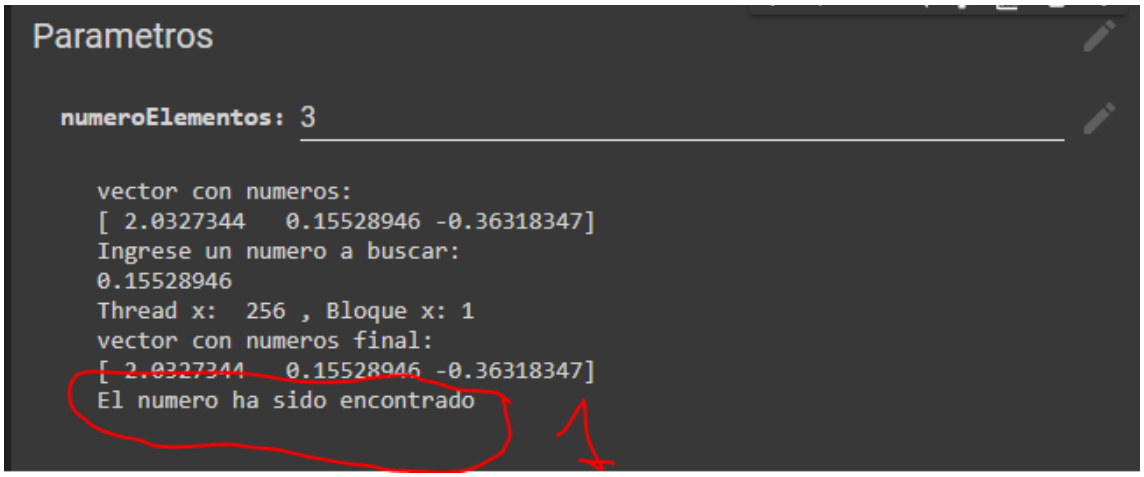
Paso3: Luego se nos solicitara por pantalla que ingresemos un numero a buscar en el vector creado previamente con números flotantes al azar, ingresamos un numero como se indica donde dice “1 (en rojo)” y después presionamos enter



Paso4a: Si el número ingresado por teclado no fue encontrado en el vector se muestra un mensaje como se indica donde dice “1 (en rojo)



Paso 4b: Si el número ingresado por teclado fue encontrado en el vector se muestra un mensaje como se indica donde dice “1 (en rojo)



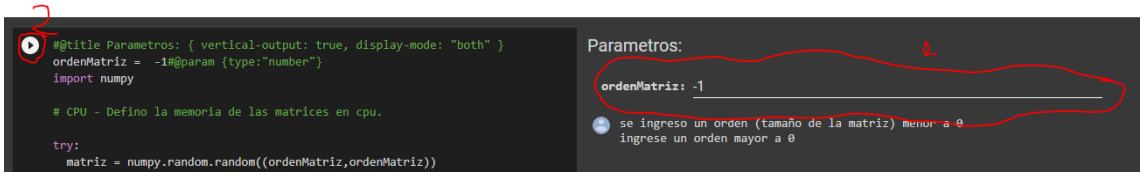
link versión con paralelismo: https://github.com/lucianopulido/EA2-Luciano-Pulido/blob/master/HPC/Pulido_Luciano_ejercicio1_1_2_GPU.ipynb

Explicación general ejercicio 2:

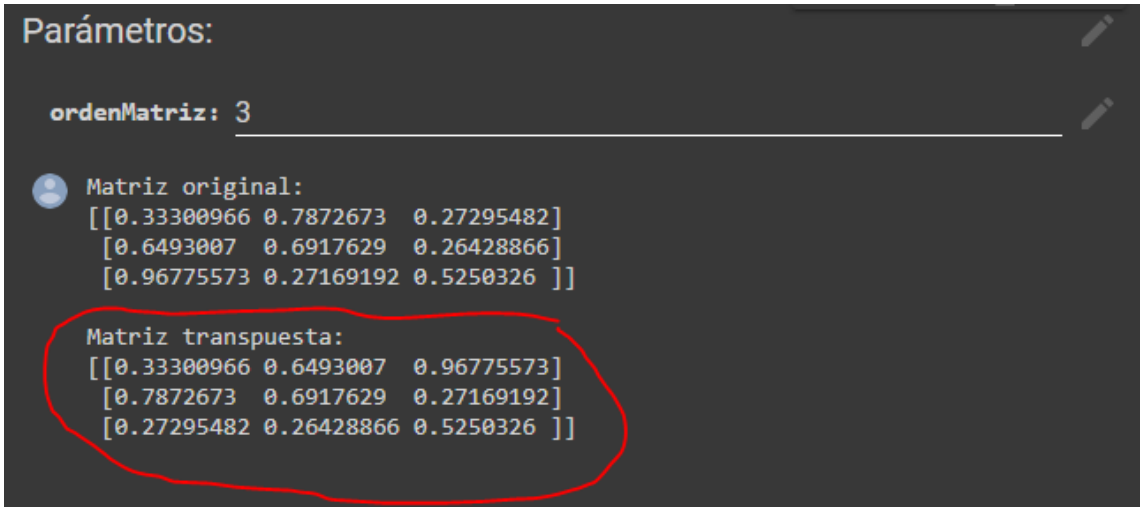
En este ejercicio se desarrolló el algoritmo para transponer una matriz tanto de manera secuencial usando solamente la CPU como usando la GPU aplicando paralelismo. En este ejercicio lo que se hace es ingresar el orden de la matriz (cantidad de filas y columnas iguales, es decir, solo admite matrices cuadradas) que deseamos que tenga una matriz y luego una vez ejecutado el algoritmo, se encarga de transponerla y nos informa la matriz transpuesta. El objetivo buscado con este ejercicio es comparar los tiempos de ejecución de la manera secuencial contra la manera que utiliza paralelismo para ver cual resulta ser más eficiente.

Manual de uso para la versión CPU (secuencial):

Paso1: Dentro del cuaderno, debemos dirigirnos a la sección “Desarrollo” y luego ingresar el orden de la matriz que deseamos que tenga la matriz que deseamos transponer, como se indica donde dice “1 (en rojo)” señalado en la imagen, luego debemos hacer click en el botón de “play” donde dice “2 (en rojo)” en la imagen.



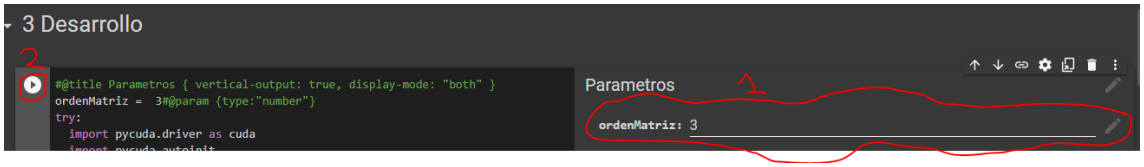
Paso2 : se informa la matriz transpuesta como en la siguiente imagen



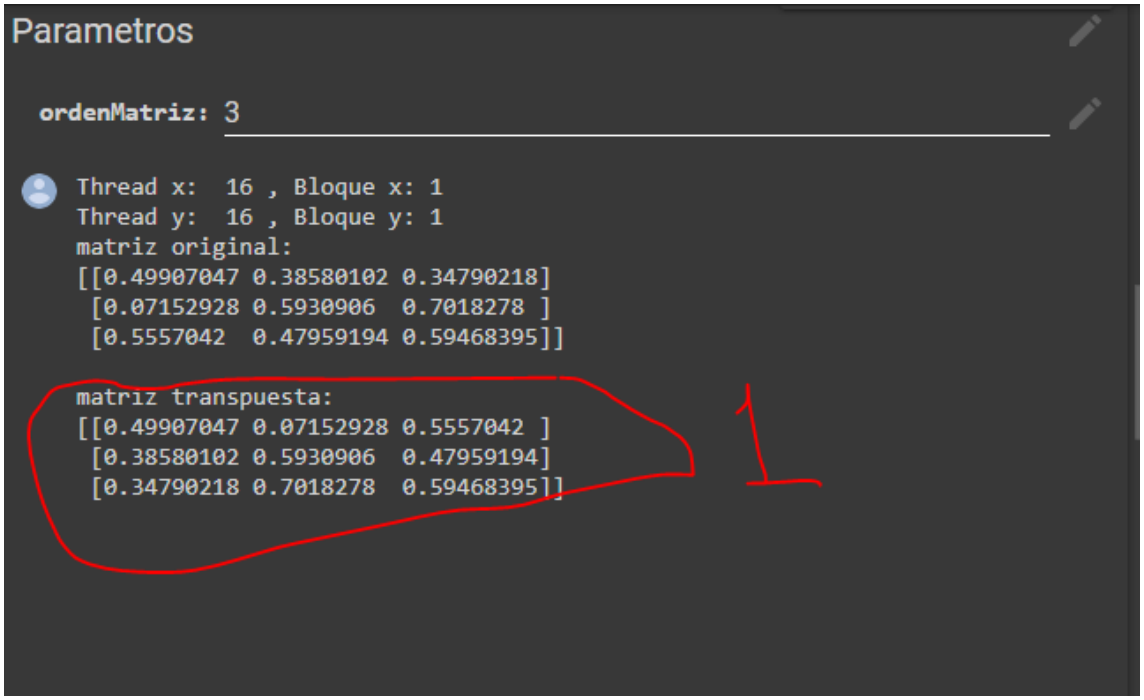
link versión secuencial: https://github.com/lucianopulido/EA2-Luciano-Pulido/blob/master/HPC/Pulido_Luciano_Ejercicio1_1_3_CPU.ipynb

Manual de uso para la versión GPU (paralelismo):

Paso1: Dentro del cuaderno, debemos dirigirnos a la sección “Desarrollo” y luego ingresar el orden de la matriz que deseamos que tenga la matriz que deseamos transponer, como se indica donde dice “1 (en rojo)” señalado en la imagen, luego debemos hacer click en el botón de “play” donde dice “2 (en rojo)” en la imagen.



Paso2: se informa la matriz transpuesta como en la siguiente imagen



Aclaración: esta versión debe ser ejecutada en el entorno de GPU.

link versión con paralelismo: https://github.com/lucianopulido/EA2-Luciano-Pulido/blob/master/HPC/Pulido_Luciano_Ejercicio1_1_3_GPU.ipynb

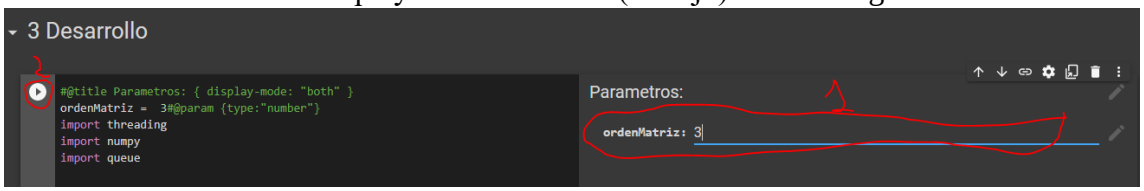
Explicación general ejercicio 3:

En este ejercicio se desarrolló el algoritmo de una suma de matrices usando solamente la CPU y aplicando paralelismo mediante Threads. En este ejercicio lo que se hace es ingresar el orden (cantidad de filas y columnas iguales, es decir, solo admite matrices cuadradas) que deseamos que tengan 2 matrices que se van a sumar y luego una vez ejecutado el algoritmo, se encarga de sumar las matrices y nos informa la matriz resultante de las sumas de las mismas. Para hacer esto, lo que hago es usar 2 hilos (Threads), donde cada uno llama a una función distinta, donde cada uno lee los números de la matriz correspondiente y los pone cada uno en una cola distinta y todo lo

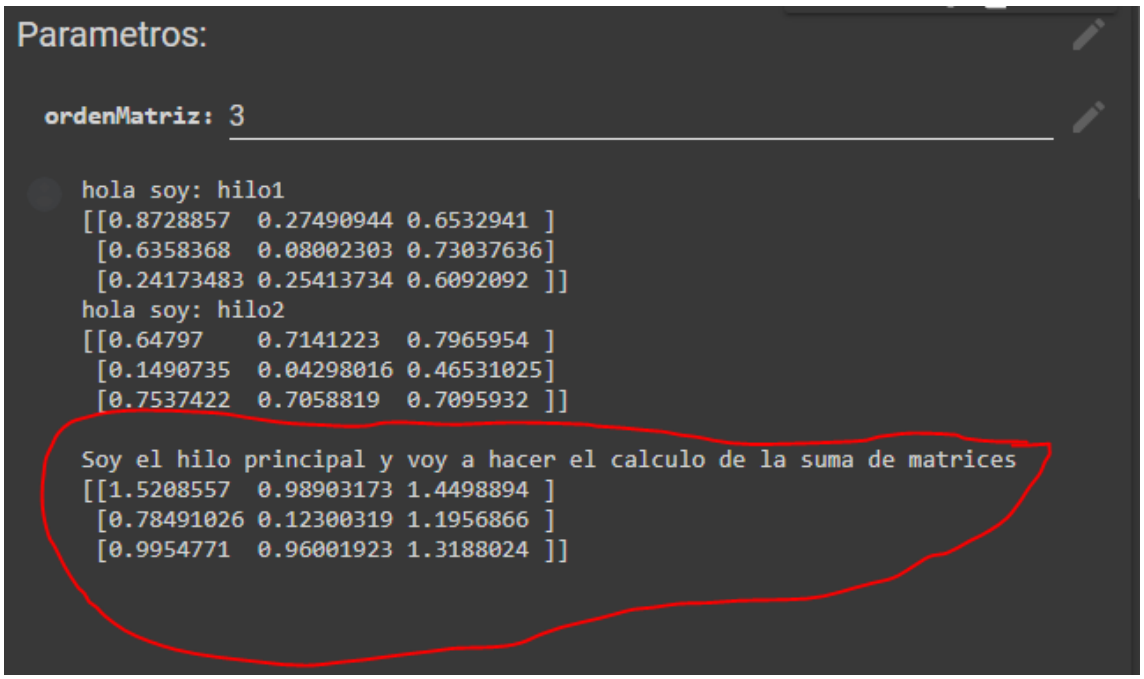
mencionado, lo hacen en paralelo. Finalmente el hilo principal saca un elemento de cada cola, los suma y los guarda en la posición correspondiente de la matriz de resultados.

Manual de uso (paralelismo):

Paso1: Dentro del cuaderno, debemos dirigirnos a la sección “Desarrollo” y luego ingresar el orden de la matriz que deseamos que tengan la matrices que deseamos sumar, como se indica donde dice “1 (en rojo)” señalado en la imagen, luego debemos hacer click en el botón de “play” donde dice “2 (en rojo)” en la imagen.



Paso2: se informa la matriz Resultante, como en la siguiente imagen de la suma de 2 matrices llenadas con números al azar y con el orden ingresado como parámetro



link versión con paralelismo: https://github.com/lucianopulido/EA2-Luciano-Pulido/commit/41df74b057f98992bbd4655a9315011bf2cb2807

Resumen general de todo el trabajo:

Respecto a los 3 ejercicios del EA3 pude sacar varias conclusiones sobre los temas vistos sobre el rendimiento de crear un programa de manera secuencial y contrastándolo con su versión con paralelismo utilizando la GPU.

En el ejercicio 1 realizamos una búsqueda secuencial, donde se busca un número ingresado por teclado dentro de un vector de números y se informa si se lo encontró o no. Los tiempos de ejecución terminaron siendo más eficientes utilizando solamente la CPU de manera secuencial porque al ser un algoritmo donde se va comparando todo el tiempo el número buscado con cada posición del vector tanto en la versión secuencial(CPU) como en la versión con paralelismo (CPU-GPU), el costo termina siendo mayor en la versión que se utiliza paralelismo con la GPU por las capas intermedias por las que deben pasar los datos desde que se envían desde la CPU a la GPU hasta que vuelven de la GPU a la CPU, le suman un tiempo mayor al propio dado por la búsqueda y por ese motivo termina siendo más eficiente realizar el algoritmo de utilizando al CPU de forma secuencial utilizando un bucle.

En el ejercicio 2 realizamos el algoritmo que obtiene la matriz transpuesta de una matriz dada. Los tiempos de ejecución en este caso resultaron más eficientes utilizando paralelismo utilizando el GPGPU porque al ser un algoritmo donde se puede mover grandes cantidades de datos accediendo a ellos y moviéndolos de manera directa por la gran cantidad de hilos que nos ofrece la GPU, esto trae como consecuencia una mayor eficiencia con tiempos de ejecución pequeños en comparación con la versión secuencial porque los tiempos que nos producen los 2 bucles de la versión secuencial son muy grandes respecto a la versión con paralelismo.

Respecto al punto 3 puedo decir que utilice paralelismo utilizando 2 Threads en Background para leer cada uno los números de la matriz que le corresponde y ponerlos cada uno en una cola distinta para comunicarse mediante esas colas con el hilo principal, el cual se encarga sacar un numero de cada cola, realizar la suma y guardarla en la posición correspondiente de la matriz de resultados de las sumas.

Promedio de tiempo total ejercicio 1 secuencial: 3.239,6 ms

Promedio de tiempo de Bucle ejercicio 1 secuencial: 0,2534 ms

Promedio de tiempo CPU ejercicio 1 paralelismo: 4.665,4 ms

Promedio de tiempo GPU ejercicio 1 paralelismo: 506,9 ms

Promedio de tiempo ejercicio 2 secuencial: 12.224,4 ms

Promedio de tiempo de Bucle ejercicio 2: 11.497,4 ms

Promedio de tiempo CPU ejercicio 2 paralelismo: 2.407 ms

Promedio de tiempo GPU ejercicio 2 paralelismo: 0,2676 ms

Autoevaluación:

Considero que logre comprender las situaciones y conceptos relacionados a la creación de una programa creado utilizando la GPU, las situaciones en donde la GPU es beneficiosa usarla y en las situaciones en las que nos perjudica . Pude notar que en los algoritmos donde hay intercambios de datos de distintas posiciones de memorias, búsquedas o hay la misma cantidad de comparaciones, utilizar la GPU nos perjudica porque terminamos teniendo un tiempo de ejecución mayor en comparación a la manera secuencial porque los datos tienen que pasar por muchas capas entre la CPU y la GPU, en cambio cuando tenemos que copiar grandes cantidades de datos es más beneficioso utilizar paralelismo con la GPU porque al acceder y copiar los datos de manera directa compensamos de manera optima el tiempo que nos lleva utilizar varios bucles en la versión secuencial. Por estamos motivos mencionados y por además cumplir los puntos extras necesarios considero que merezco una nota de 8.