Universidad de la República
Facultad de Ingeniería, INCO
Montevideo, Uruguay.

Práctico 5:

GPGPU

Grupo 10

Integrantes:

Federico Gil - 5.198.750-6 - federico.gil@fing.edu.uy

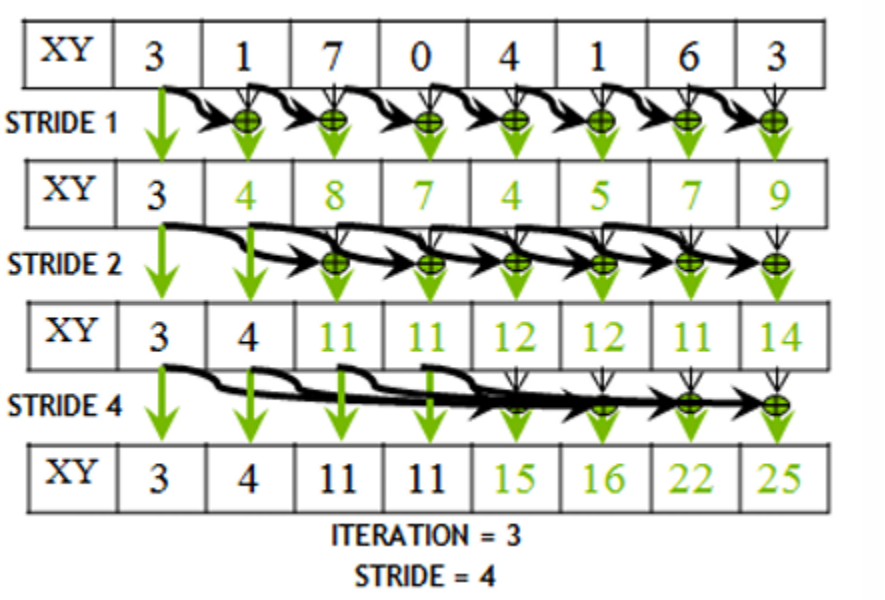
Diego Amorena - 5.011.502-7 - diegoamorenag@gmail.com

Ejercicio 1

Parte A

En este ejercicio, se desarrolló un algoritmo de scan que utiliza el concepto de stride, para determinar las interacciones entre hilos y los datos que procesan. El stride representa la distancia entre los elementos que cada hilo debe sumar(en este caso donde se utilizó la operación de suma para combinarlos). Este proceso comienza con un stride de uno, que se incrementa, doblando su valor en cada iteración sucesiva del algoritmo.

Concretamente, cada hilo es responsable de sumar un par de elementos: uno en la posición que corresponde directamente a su identificador de hilo y otro que se encuentra a una distancia definida por el valor actual del stride. A medida que el algoritmo avanza, el stride se amplía, obligando a cada hilo a extender el rango de su cálculo para incluir componentes progresivamente más alejados dentro del arreglo. Este procedimiento se repite hasta que se ha realizado un scan completo del arreglo, asegurando que todos los elementos necesarios han sido adecuadamente sumados según el patrón definido por el stride.



Una limitación de este algoritmo es la desigual distribución de carga entre los hilos durante su ejecución, resultando en un desequilibrio significativo en la cantidad de procesamiento requerido por cada uno. Por ejemplo, el hilo con un thread id 0 no realiza cálculos, limitándose a escribir en el índice 0 del arreglo de salida, mientras que el hilo cuyo

thread id es n (siendo n la cantidad total de elementos en el arreglo) puede realizar hasta log n iteraciones. Esta disparidad se manifiesta particularmente hacia las etapas finales del algoritmo, donde los hilos con identificadores más altos deben de realizar todo el cómputo restante, en contraste con aquellos hilos con identificadores más bajos, ya completaron sus asignaciones. Esta variabilidad en la carga de trabajo puede llevar a una utilización ineficiente de los recursos de la GPU, ya que algunos hilos quedan inactivos mientras otros aún están procesando activamente.

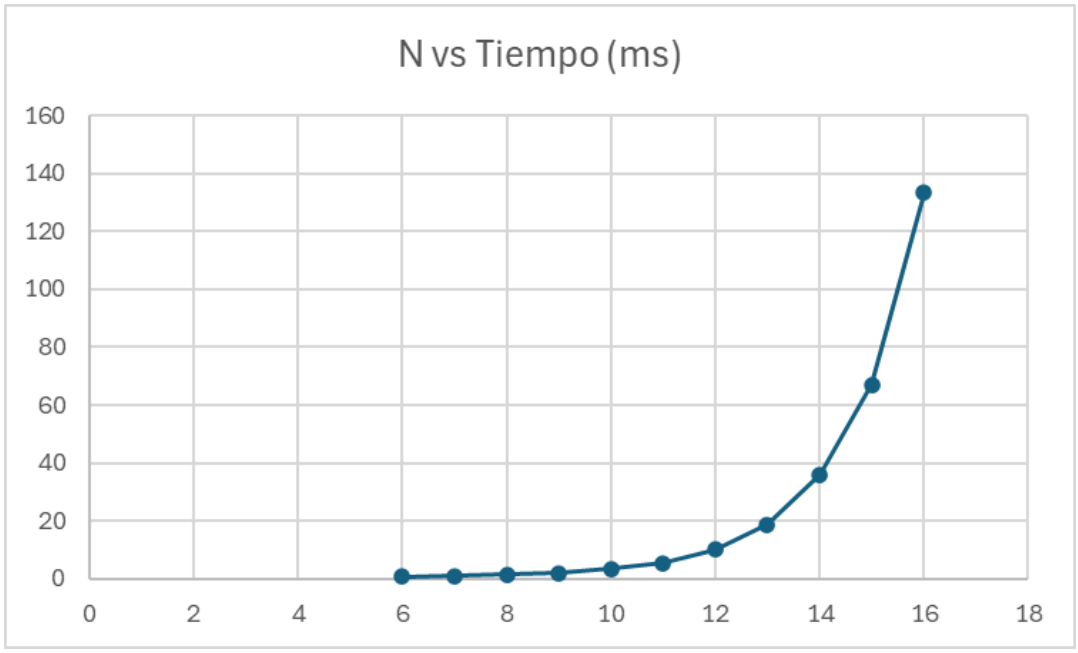
Otra desventaja es la sincronización entre hilos y bloques que requiere. Los hilos deben leer dos de los componentes del arreglo de lectura y escribir en el arreglo de salida en cada una de las iteraciones. Los hilos compiten por el acceso al mismo espacio de memoria para la lectura de los componentes, por lo que ocurre un conflicto de banco.

A la hora de evaluar el desempeño del algoritmo en la GPU, se observó que la primera ejecución resultaba considerablemente menos performante que las ejecuciones siguientes. Se atribuyó esto principalmente a la inicialización del contexto de la GPU, que implica la configuración del entorno de ejecución necesario para el kernel. Para obtener medidas más precisas del rendimiento real del algoritmo, se decidió hacer una ronda de calentamiento previo a comenzar a medir y registrar los tiempos de ejecución. Este enfoque asegura que las métricas de desempeño reflejan el comportamiento más estable y representativo del algoritmo.

En la siguiente imagen se aprecia el tiempo medio de ejecución y desviación estándar para 10 muestras en cada valor de N.

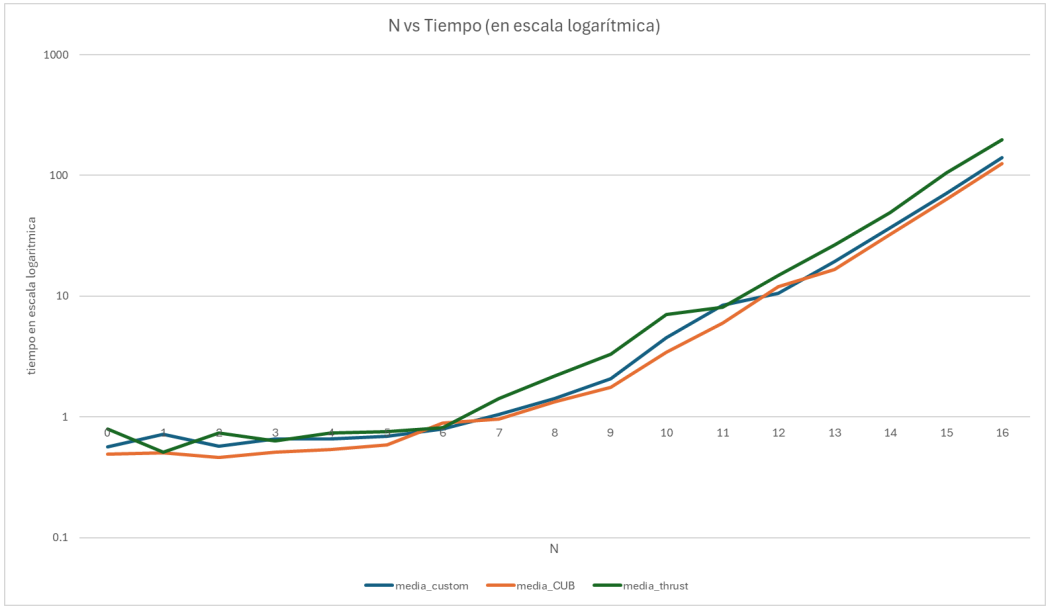
```
N = 6 -> Iguales        Media de tiempo: 0.73042 ms     Desviacion: 0.108126 ms
N = 7 -> Iguales        Media de tiempo: 0.90082 ms     Desviacion: 0.183042 ms
N = 8 -> Iguales        Media de tiempo: 1.48745 ms     Desviacion: 0.349916 ms
N = 9 -> Iguales        Media de tiempo: 1.97419 ms     Desviacion: 0.208495 ms
N = 10 -> Iguales       Media de tiempo: 3.55234 ms     Desviacion: 0.546709 ms
N = 11 -> Iguales       Media de tiempo: 5.51704 ms     Desviacion: 0.241844 ms
N = 12 -> Iguales       Media de tiempo: 10.0061 ms     Desviacion: 0.736699 ms
N = 13 -> Iguales       Media de tiempo: 18.6483 ms     Desviacion: 0.611702 ms
N = 14 -> Iguales       Media de tiempo: 35.9925 ms     Desviacion: 1.05656 ms
N = 15 -> Iguales       Media de tiempo: 67.0707 ms     Desviacion: 1.46961 ms
N = 16 -> Iguales       Media de tiempo: 133.352 ms     Desviacion: 1.68873 ms
```

Asimismo, la siguiente imagen muestra como varía el tiempo de ejecución según N de manera más visual. Medido en milisegundos.



Parte B

En esta sección se comparan los tiempos de ejecucion tambien para 10 experimentos por cada N de las implementaciones usando las librerías CUB y Thrust



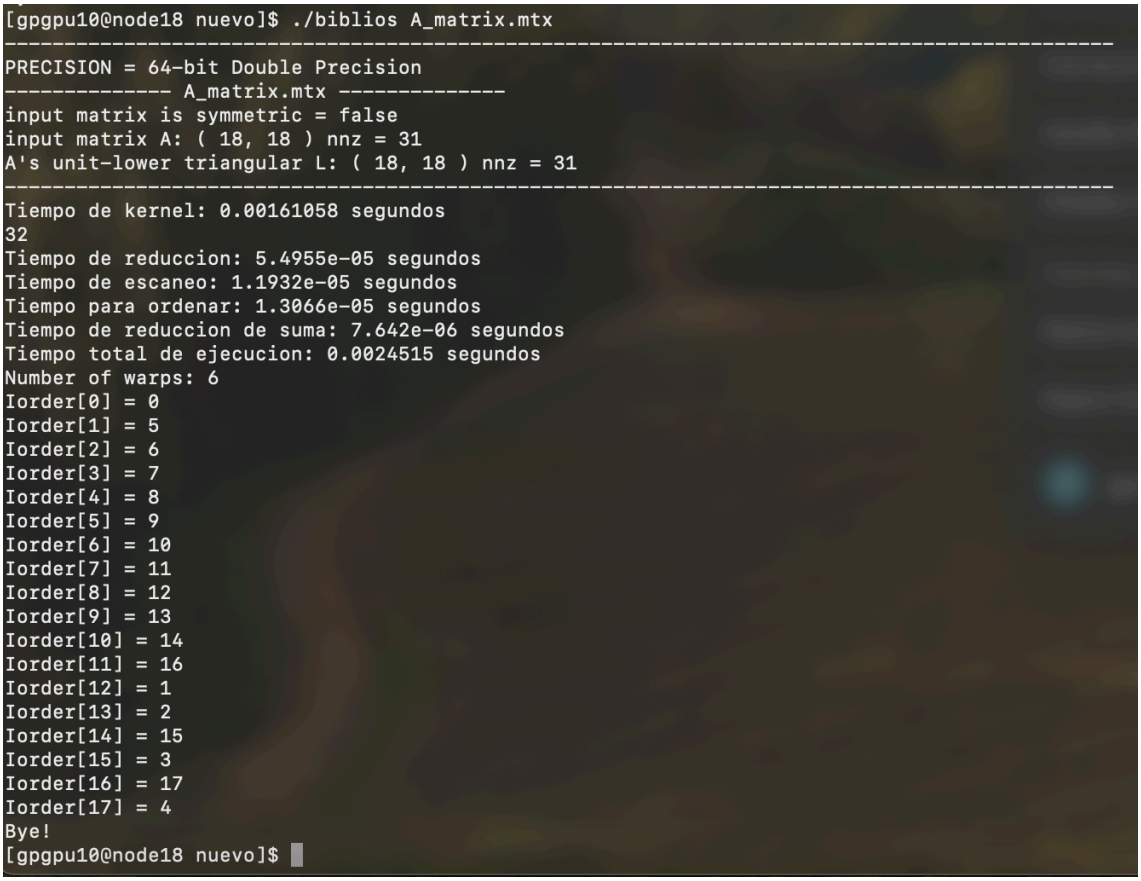
Para hacer más fácil el análisis se tomó el eje vertical (de tiempo) en escala logarítmica. Lo más interesante a notar es que Thrust es consistentemente peor que nuestra implementación,

mientras que CUB consistentemente mejor. Estamos satisfechos con que nuestra implementación se encuentre entre estas dos grandes librerías, esto muestra el buen uso de técnicas de optimización.

Ejercicio 2

Para resolver este ejercicio se utilizaron tres structs que proveyeron funciones para obtener el nivel, mapear al rango pedido y una última para asignar al warp. Dentro de ordenar filas, se obtuvo el resultado del kernel como antes, se aplicaron reducciones y las transformaciones antedichas, se generaron los histogramas utilizando CUB así como los Exclusive scan de la misma libreria. Finalmente se hicieron los *sortings* usando RadixSort y la asignación a warps. Para cada etapa relevante se midieron los tiempos medios de cada una de las matrices. Para la matriz de ejemplo A_matrix.mtx se obtuvo un tiempo medio de 0.2496 ms y desvio de 0.00559 ms levemente peor que la implementación brindada para CPU en la cual la media fue de 0.2024 ms y desvío de 0.005265 ms.

La siguiente imagen muestra el resultado de la ejecución de nuestra implementación



Cabe aclarar que estos valores tanto del número de warps como la asignación son consistentes con la implementación en CPU brindada.

La siguiente tabla resume cómo se comporta nuestra implementación comparada con la de gpu en tiempo total de ejecución:

Resumen	CPU		GPU		Pertenencia al IDC
Matriz	Media	Desvio	Media	Desvio	
A_matrix	0.2025	0.0053	0.2496	0.0056	No pertenece al IDC
1	0.7361	0.0544	0.8210	0.0252	No pertenece al IDC
2	0.7189	0.0595	0.7877	0.0284	No pertenece al IDC
3	1.0098	0.3313	0.8534	0.0608	Pertenece al IDC
4	0.8943	0.0203	0.9676	0.0035	No pertenece al IDC
5	0.3297	0.1968	0.4595	0.2916	Pertenece al IDC

Se puede apreciar en la tabla que en la mayoría de las matrices la implementación secuencial fue mejor que nuestra implementación paralela.

Para más nivel de detalle en particular en la parte paralelizada, se muestra la siguiente tabla para las mismas matrices.

Parte paralela vs no paralela (en milisegundos)					
Resumen	CPU		GPU		Pertenencia al IDC
Matriz	Media	Desvio	Media	Desvio	
A_matrix	0.0072	0.0007	0.0545	0.0009	No pertenece al IDC
1	0.0216	0.0004	0.1204	0.0070	No pertenece al IDC
2	0.0229	0.0003	0.1328	0.0148	No pertenece al IDC
3	0.0694	0.0996	0.1282	0.0103	Pertenece al IDC
4	0.0209	0.0065	0.1066	0.0018	No pertenece al IDC
5	0.0430	0.0047	0.0075	0.0075	No pertenece al IDC

Desafortunadamente los resultados no son alentadores, nuestra implementación es peor en cuanto a tiempos que la secuencial. Esto se puede deber a muchos motivos, por ejemplo el

tamaño de la matriz, para matrices pequeñas la diferencia es más notoria, pero para matrices más grandes como es el caso de la matriz 3 se puede apreciar que la diferencia disminuye e incluso está dentro del margen de error de la otra implementación.

Creemos que la diferencia en los tiempos se debe a el tiempo de inicialización que requiere la GPU que se vería diluido en matrices más grandes.

Todas las matrices estaban compuestas de enteros en doble precisión. A continuación se listan el tamaño de cada una de ellas, la cantidad de elementos no ceros.

Número de Matriz	Simetría de la Matriz	Dimensiones de la matriz	Número de Elementos No Ceros en A
A_matrix	No	18 x 18	31
1	Sí	8000 x 8000	32000
2	Sí	9000 x 9000	36000
3	Sí	10000 x 10000	40000
4	Sí	2000 x 2000	41906
5	No	11380 x 11380	39206

Todas las pruebas se ejecutaron en clusterUY, en el nodo 18, el cual cuenta con una GPU de Nvidia p100.