Multiplicación Matriz Vector

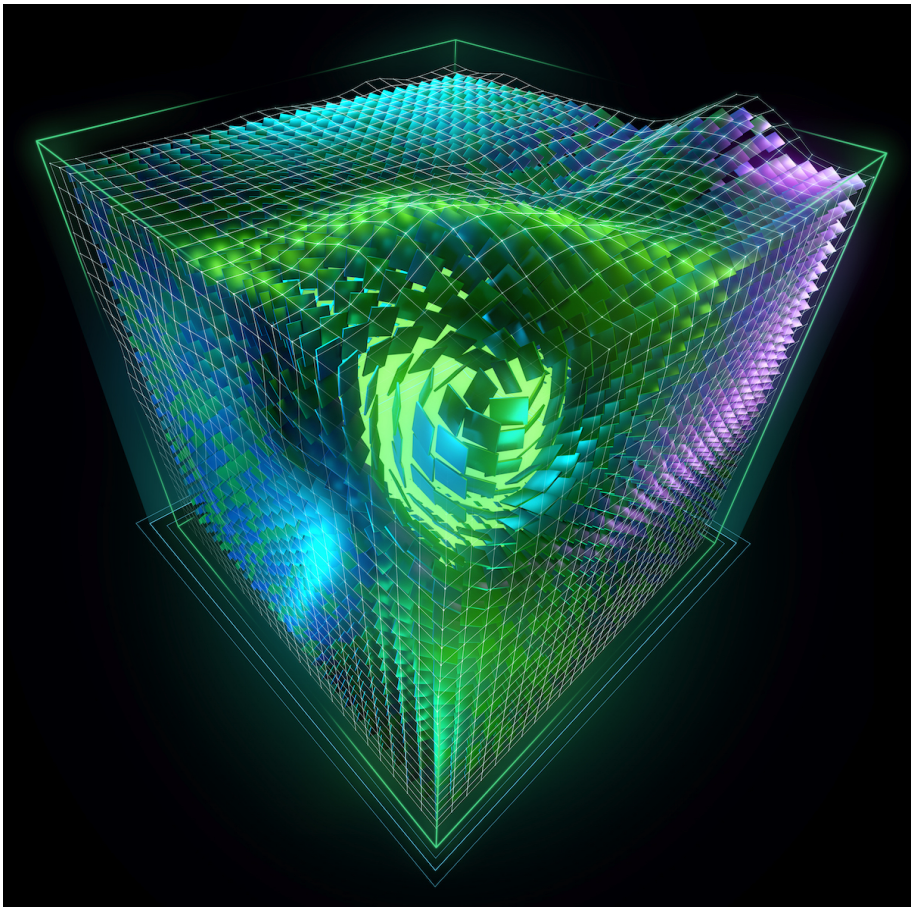
Laboratorio 3

Ignacio Aedo - 201773556-2
Rodrigo Cayazaya - 201773538-4
Danny Fuentes - 201773559-7
Jean-Franco Zárate - 201773524-4

Características GPU

Collab:

- Modelo: Nvidia Tesla T4
- Compute capability: 7.5



Preguntas:

1.

Los tiempos obtenidos por cada implementación se muestran en la Tabla 1.

Incisos	Tiempo GPU en ms
kernelA	144.455643
kernelx	8.685248
kernelb	3.787040
kernelRed	9.739296
kernelSM	3.803136
kernelCM	3.61062

Tabla 1: Tiempos obtenidos por cada implementación realizada

Es posible notar que la implementación realizada en el kernelCM es la más rápida con aproximadamente 3.6 [ms]. Esto resulta una mejora sustancial frente al método inicial (kernelA), el cual presenta un tiempo aproximado de 144.5 [ms]. Además de presentar el mejor tiempo, la implementación realizada es una de las más simples para el programador, lo cual significa grandes ventajas a la hora de realizar una operación tan común en diferentes áreas como la multiplicación matriz-vector. En cuanto a los demás métodos, el kernelb obtiene un tiempo un relativamente mayor con un enfoque casi idéntico, mientras que el resto se alejan tanto en tiempo como en dificultad de implementación.

2.

El kernelCM presenta el mejor tiempo, lo cual se debe a la utilización de memoria constante. Si bien esta memoria tiene tiempos de acceso similares a la memoria global, nos vemos beneficiados por el uso de *constant cache*, el cual presenta latencias similares a los registros. Esto combinado con el acceso repetido de la variable almacenada en memoria constante y el tamaño relativamente pequeño del vector *x* logran explicar de buena forma el resultado obtenido. Sin embargo, la memoria constante tiene una capacidad limitada de 64KB y su uso presenta ventajas solo en momentos específicos.

Por otro lado, en el resto de kernel se notó la latencia introducida por el uso de memoria global y las operaciones atómicas que, si bien nos ayudan a evitar condiciones de carrera, aumentan (aunque en ciertas ocasiones es muy poco) el tiempo de ejecución.

Llama la atención el tiempo obtenido con el kernelRed y el kernelSM, los cuales hacen uso de la memoria compartida, la cual es la más rápida después de los registros. Esto se puede explicar en parte por la necesidad de sincronización de las hebras. En este punto se puede reflexionar sobre la importancia de la implementación, el cual puede ser más relevante que la memoria utilizada.

Conclusión

A modo de conclusión, se dio cuenta de la importancia del contexto de trabajo y la implementación a utilizar, ya que pueden tener un peso mucho mayor que solo usar un tipo de memoria porque es la más rápida teóricamente. De esta forma, entender el contexto y aplicar la implementación adecuada sumado al uso de la memoria indicada resulta en mejoras importantes de rendimiento. Por otro lado, los resultados obtenidos están condicionados en gran medida por el tamaño del vector *x,* por lo que resultaría un experimento interesante para trabajos futuros comparar el desempeño de las distintas implementaciones y memorias con mayores tamaños, considerando el espacio limitado de la memoria constante.