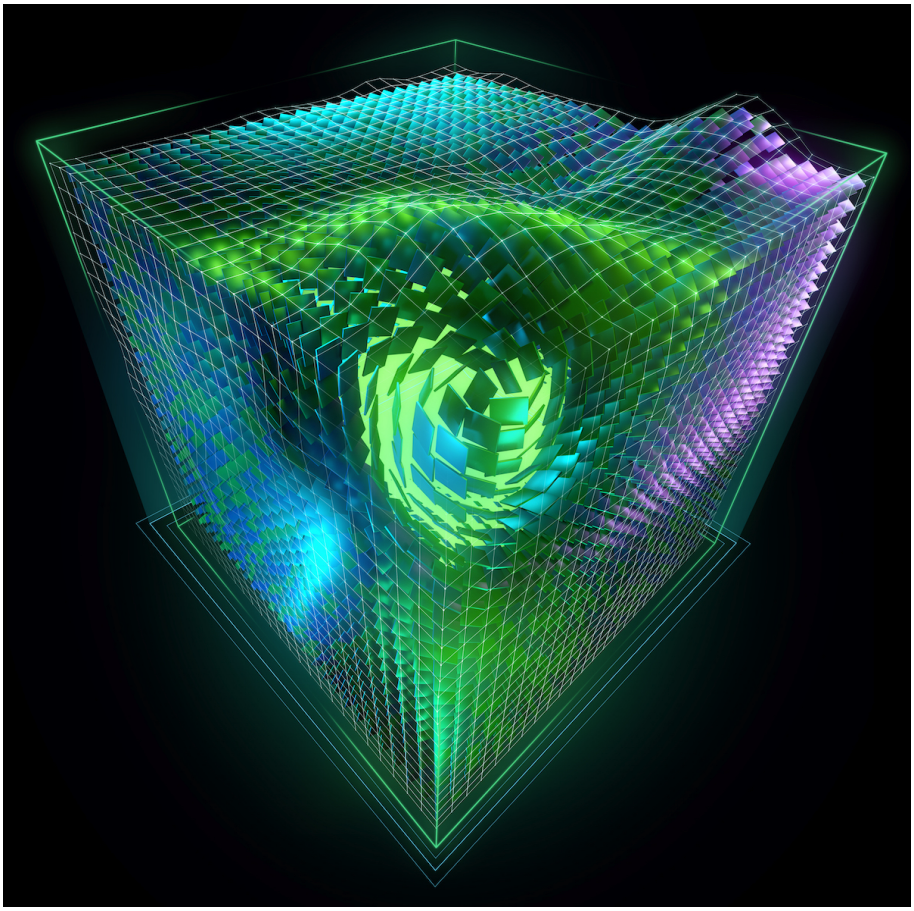
Streams

Laboratorio 4

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

Características GPU

- Modelo: GTX 1660 Super
- Compute capability: 7.5



Preguntas:

[Informe]

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

Kernels	Tiempo GPU en milisegundos
preguntaUno	10237
preguntaDos	657.99
preguntaTres	557.73
preguntaCuatro	1711.47

Tabla 1: Tiempos obtenidos por cada implementación realizada

Es posible notar que el mejor tiempo lo obtiene el kernel preguntaTres, mientras que el peor tiempo es obtenido por la implementación preguntaUno.

[Pregunta] Comente sobre los tiempos observados y argumente sobre por qué cada implementación es más o menos eficiente que las demás

La implementación preguntaUno muestra el peor tiempo con diferencia, lo cual es un resultado esperado ya que la CPU debe actualizar de forma secuencial *MxN* datos un total de *t* veces. La implementación que menos demora corresponde al kernel preguntaTres, lo cual también corresponde a un resultado esperado, ya que aprovechamos al máximo la simultaneidad de la GPU al utilizar CUDA streams a diferencia del kernel preguntaDos que utiliza solo el stream 0. Al utilizar CUDA streams podemos dividir la carga de la GPU en distintos kernels que trabajan de forma simultánea, haciendo la implementación más eficiente. Por otro lado, la implementación realizada en preguntaCuatro presenta pasos extras que agregan complejidad algorítmica gracias a la distribución de los datos, lo cual causa un aumento de tiempo importante.

[Pregunta] ¿Qué desafíos enfrentó al realizar la última implementación? ¿Explique detalladamente cómo afrontó estos desafíos y por qué eligió dicha solución?

El principal desafío que enfrentamos fue la compleja distribución de los datos en los diferentes streams, ya que la actualización de los datos depende de estos vecinos contiguos. Debido a que se necesitó acceso a los datos pertenecientes a los diferentes streams, se utilizaron dos matrices ubicadas en memoria global de CUDA. La primera matriz se utiliza para calcular el valor de la iteración *t,* mientras que la segunda es utilizada para guardar el valor calculado. Luego, se ejecutaron *t* veces los 4 streams definidos, donde cada hebra se encargó de calcular y copiar cada dato de toda la fila, para finalmente copiar de vuelta a la matriz original.

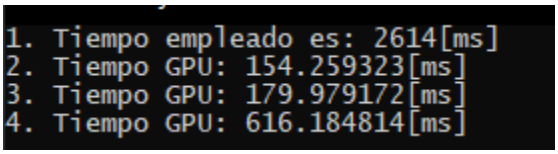
Se eligió esta solución debido a que es la forma más simple de implementar, aunque presenta mayores tiempos que el resto de kernel y ocupa gran cantidad de memoria.

Conclusión

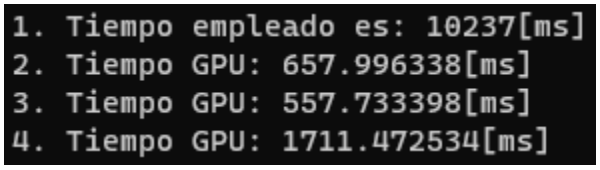
A modo de conclusión, notamos la relevancia que tiene la utilización de streams en el procesamiento de grandes volúmenes de datos, puesto que nos permite dosificar el trabajo realizado por los diferentes kernel implementados. Sin embargo, es importante tener en cuenta la correcta distribución de los datos, puesto que dividir el trabajo en streams con una estructura de datos mal planteada puede dirigir a una disminución de la eficiencia y un aumento de la dificultad de implementación.

*Nota

La GPU utilizada obtuvo los tiempos mostrados en la tabla 1, pero probamos con las tarjetas RTX 3060 TI y la Tesla T4 de Google Colab y el kernel preguntaDos obtuvo tiempos ligeramente mejores que el kernel preguntaTres.



Tiempos de la RTX 3060 TI



Tiempos de la GTX 1660 Super