
*Departamento de Ingeniería Informática, Universidad Técnica Federico Santa María*

# Laboratorio 2
# "Intercalado Coalescente de Columnas"

| | |
|---|---|
| Asignatura: | Computación de alto desempeño |
| Profesor: | Álvaro Salinas |
| Alumnos: | Camilo Maldonado 201573013-k<br>Diego Córdova 201403009-6 |

6 de Mayo del Año 2019, Valparaíso, Chile.

# Introducción

En el siguiente laboratorio se evalúan conocimientos sobre divergencia, uso de memoria global y accesos de memoria coalescentes. Adicionalmente, se usa la herramiento Visual Profiler para analizar el rendimiento de diferentes algoritmos GPU.

# Especificaciones técnicas

Las especificaciones del computador y tarjeta gráfica usada.

- Tarjeta gráfica Nvidia GeForce GTX 560, Compute Capability 2.1
- Procesador intel i5-3550 CPU, 3.300 GHz
- 8 GB de RAM
- Windows 10, 64 bits.

# Desarrollo

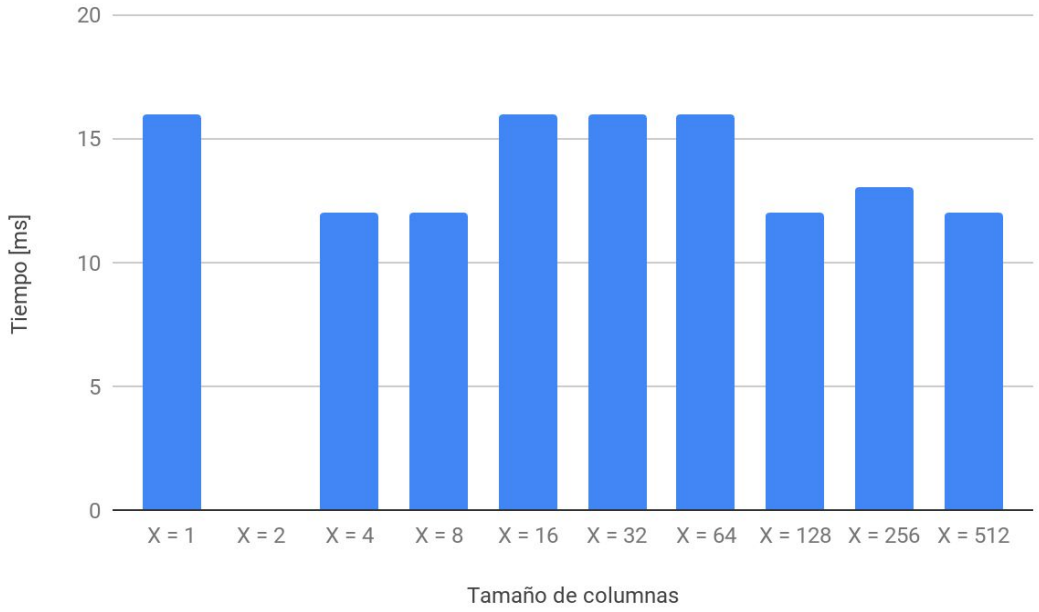
## Pregunta 1

Aquí se implementó la solución más simple posible, generando dos imágenes donde una sería para escribir y otra para leer, además de ir recorriendo uno por uno los canales e insertando los pixeles en un array resultante donde correspondan según el valor X Con la siguiente fórmula :

```
out[i] = ((i/x)%2 == 0)? in[i +x]: in[i-x];
```

Tiempos de lo que tardó el algoritmo CPU en hacer el trabajo. (Por alguna razón no imprimió los decimales a pesar de que se seteo una precisión de 4 decimales)

| Tamaño de columnas | Tiempo en [ms] |
|---|---|
| X = 1 | 16 |
| X = 2 | 0 |
| X = 4 | 12 |
| X = 8 | 12 |
| X = 16 | 16 |
| X = 32 | 16 |
| X = 64 | 16 |
| X = 128 | 12 |
| X = 256 | 13 |
| X = 512 | 12 |

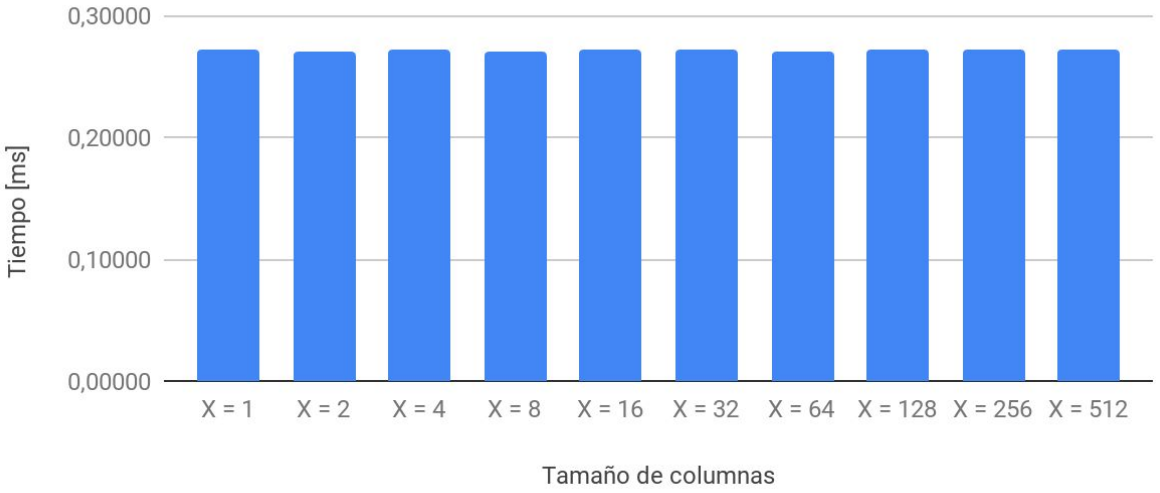



## Pregunta 2

En esta implementación se optó por la misma idea de tratar el arreglo completo para los canales y dos imágenes, una para leer y otra para escribir. Pero se trabaja con 4 bloques de 256 hebras cada uno para iterar por filas, teniendo MxN / 256 número de bloques. Donde cada thread opera 3 veces, una por cada canal y en esta operación lee de una imagen y escribe en otra, generando lectura en dos palabras distintas y como son 3 canales se tiene una lectura en 6 palabras. Otra implementación más complicada sería la de ocupar una imagen y preocuparse de las situaciones de carrera, además de preocuparse que las dos o seis lecturas, si es posible dejarlas en una misma palabra, pero para ello debería estar preprocesada las imágenes, puesta de una manera conveniente.

| Tamaño de columnas | Tiempo en [ms] |
|---|---|
| X = 1 | 0,27136 |
| X = 2 | 0,270592 |
| X = 4 | 0,271936 |
| X = 8 | 0,270976 |

| | |
|---|---|
| X = 16 | 0,271584 |
| X = 32 | 0,27216 |
| X = 64 | 0,270528 |
| X = 128 | 0,27264 |
| X = 256 | 0,271744 |
| X = 512 | 0,272288 |




## Pregunta 3

En esta implementación se agrega la condición que los primeros dos bloques de los 4 encargados por fila sean de escritura de movimientos a la derecha y los dos últimos hacia la izquierda, esto implica una condición que nos produjo un empeoramiento en el tiempo de ejecución dado que deben los threads esperar por los otros bloques, es decir corren primero todos los bloques izquierdos y luego los derechos. Además nos ocurrió un problema que desde cierto punto empieza a fallar y salirse de la memoria, intentando arreglarlo y buscando por internet creemos que es debido al compute capability deprecado de nuestra máquina y con valores grandes puede incurrir en algún desbordamiento cuando tiene una gran cantidad de threads esperando en los condicionales.

| Tamaño de columnas | Tiempo en [ms] |
|---|---|
| X = 1 | 0,56384 |
| X = 2 | 0,564064 |
| X = 4 | 0,563456 |
| X = 8 | 0,5656 |
| X = 16 | 0,672544 |
| X = 32 | 1,38368 |
| X = 64 | 1,46058 |
| X = 128 | 1,46058 |
| X = 256 | 1,46058 |
| X = 512 | 1,46058 |

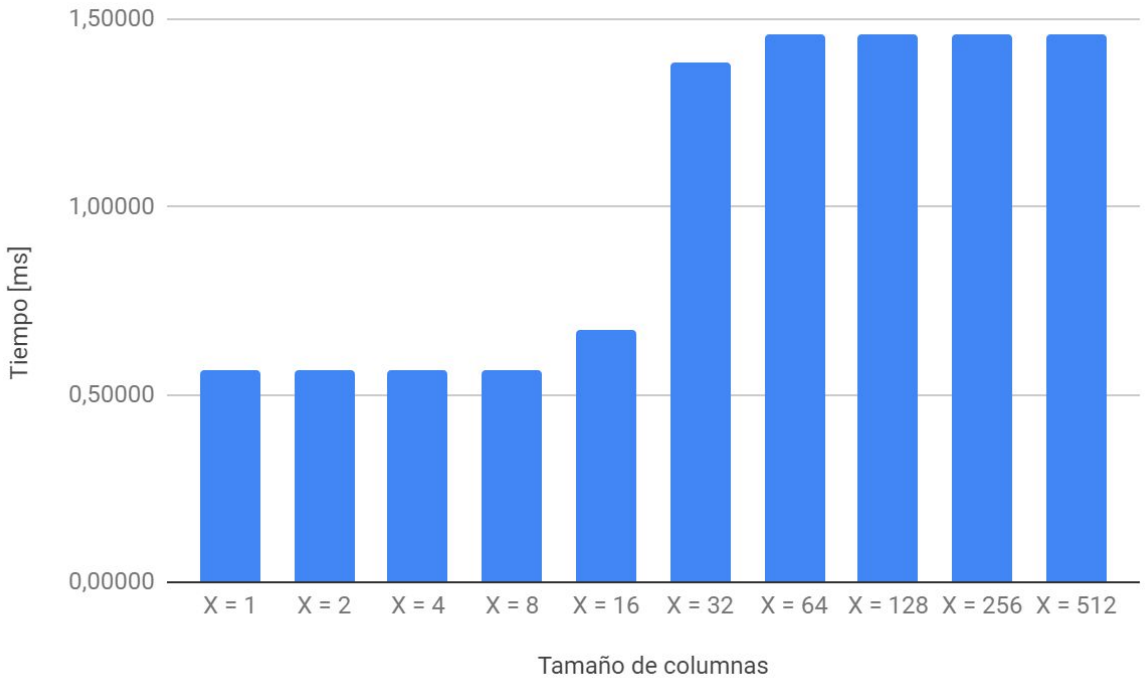
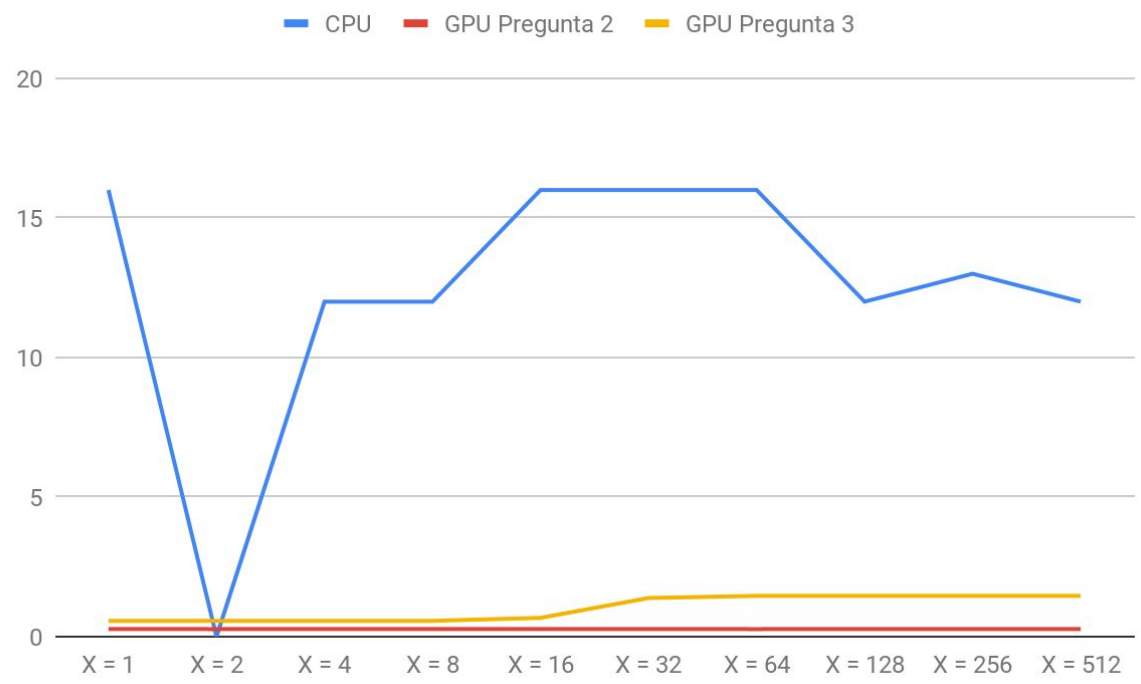
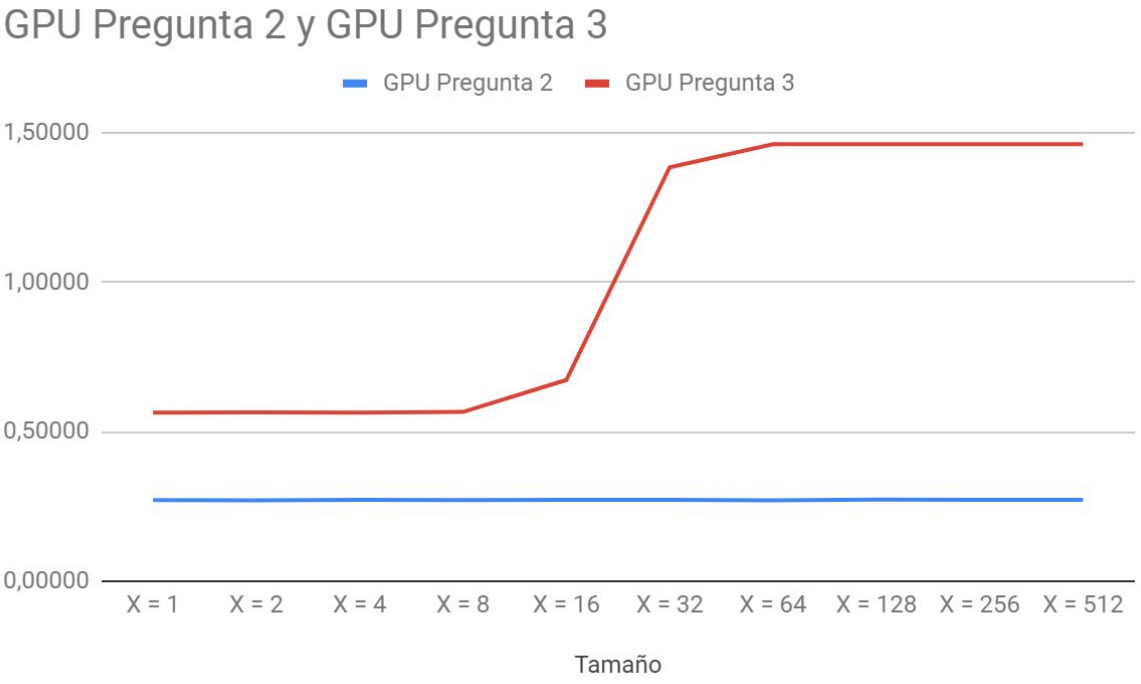


Gráfico preguntas 1, 2 y 3 juntos



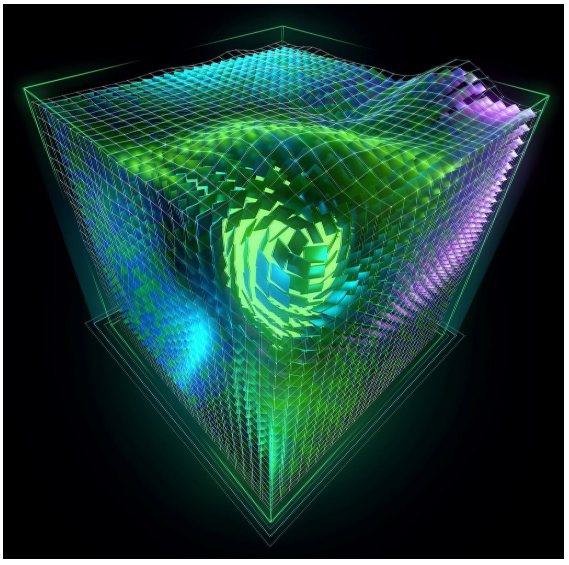
## Pregunta 4

En esta implementación no alcanzamos a hacerla, solo pensar algunas posibles soluciones pero no convencidos que fueran correctamente usada la memoria coalescente. Poniendo énfasis en el preprocesamiento al considerar que cada thread accedía a unas 6 palabras con las 2 imágenes, se podría guardar los datos con los X ya sabidos para generar un arreglo donde su largo sería 6*(MxN) ya que por cada pixel RGB se tendría RR'GG'BB' siendo los R' los valores que se moverán hacia R de acuerdo a lo impar o par de la subimagen que pertenece el píxel. Generando que las 6 lecturas que hacían los algoritmos de arriba, ahora lo hagan en una misma palabra.

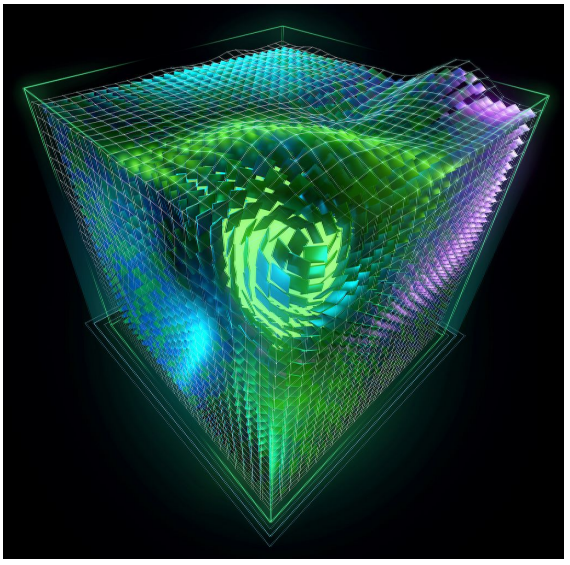
## Comparación de Banthwith

## Resultados de imágenes

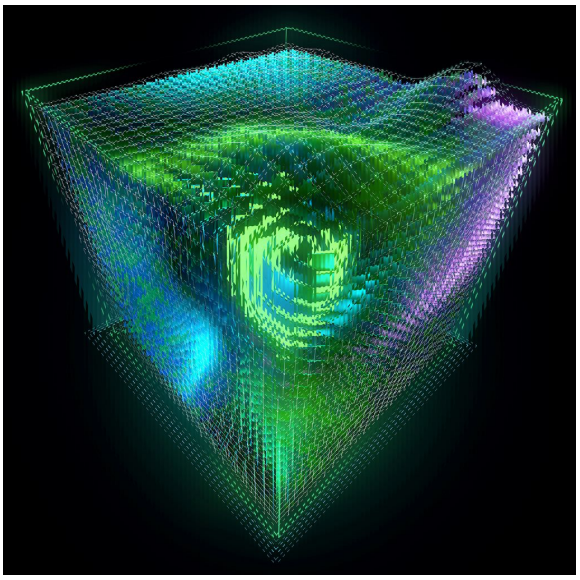
Estos son los resultados obtenidos en CPU, que fueron iguales a los GPU de la pregunta 2, pero difirieron de los de la pregunta 3 por alguna razón de implementación



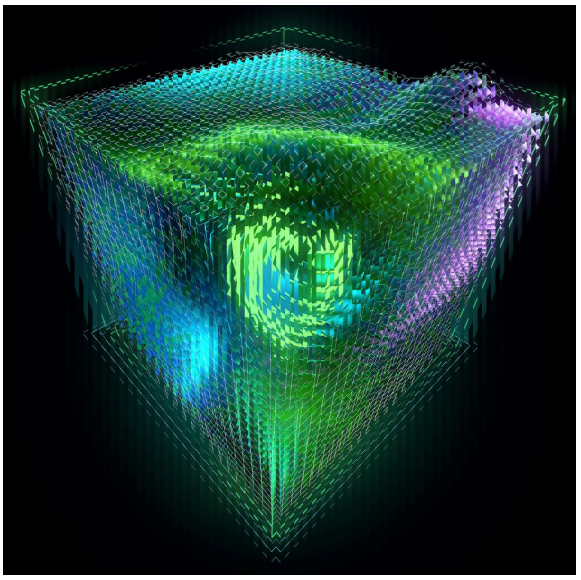
X = 1



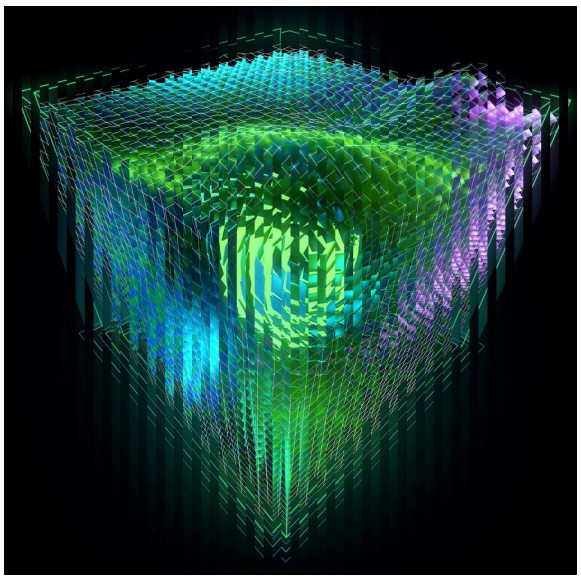
X = 2



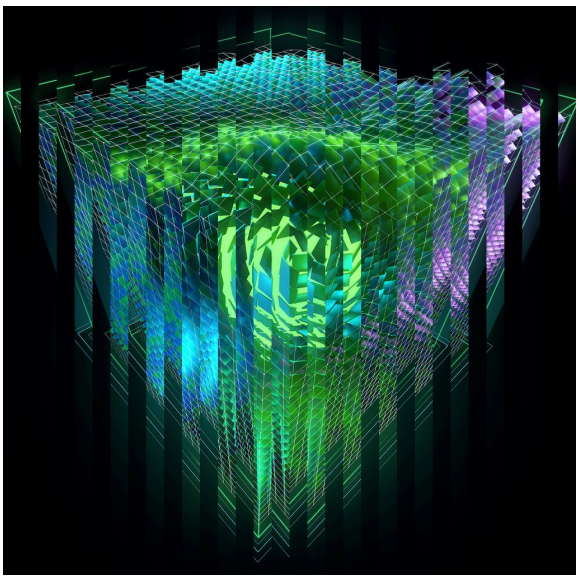
X = 4



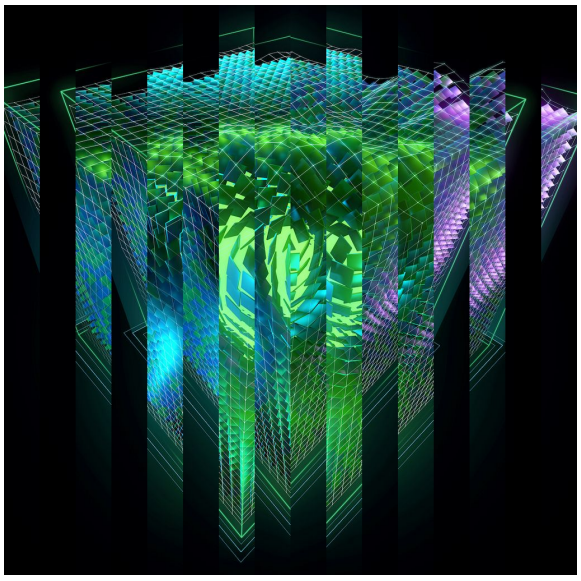
X = 8



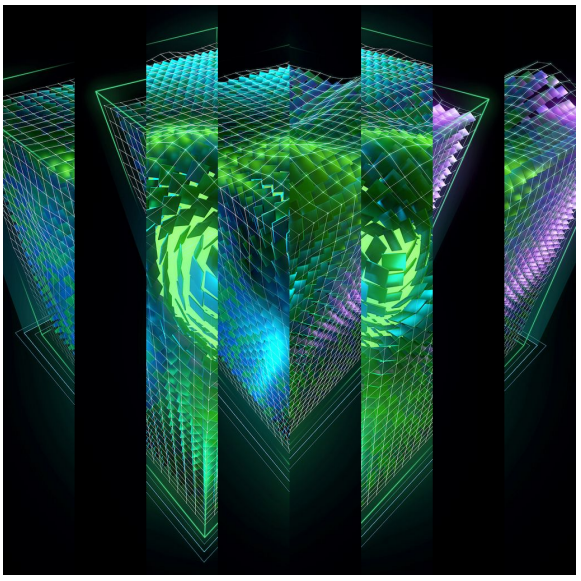
X = 16



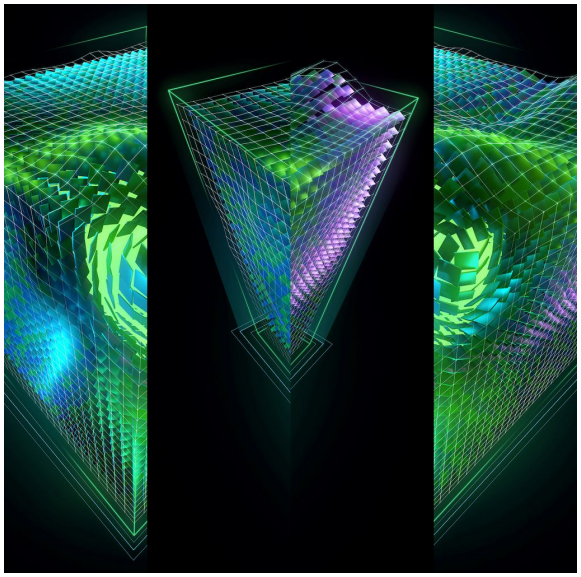
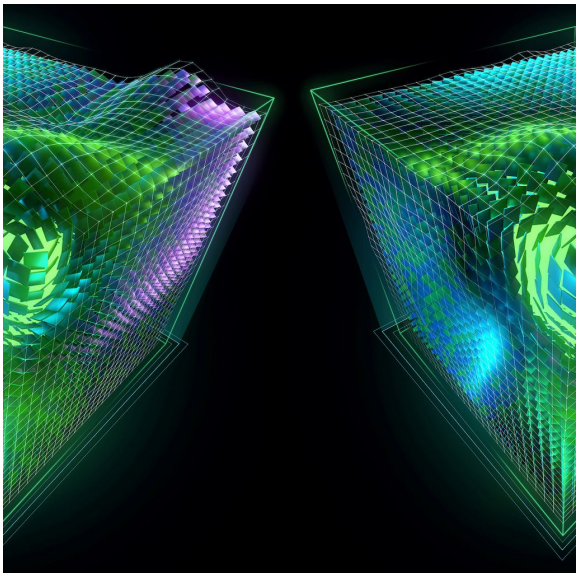
X = 32



X = 64



X = 128

| | |
|---|---|
|  |  |
| X = 256 | X = 512 |

## Conclusiones

Se notó claramente la eficiencia de GPU sobre GPU pero a la vez en la implementación fue mucho más costosa en tiempo de desarrollo y bastante más complicada de pensar. Por otro lado entre las implementaciones GPU se nota la importancia del orden de los datos y el orden de ejecución que reciben los threads o bloques, además del error que tuvimos por sobre ocupar algún elemento en los condicionales en la pregunta 3 (es nuestra suposición).