CC7515-1 — Computación en GPU
Profesor: Nancy Hitschfeld K.
Auxiliares: Diego García y Vicente González
Estudiante: Andrés Calderón Guardia

Tarea 2 - OpenCL y CUDA

El juego de la vida de Conway

1. Introducción

El juego de la vida de Conway consiste en un autómata celular bidimensional, donde cada celda puede estar viva o muerta en base a una serie de reglas definidas según el estado de la vecindad, en este caso, las 8 celdas adyacentes.

Para esta tarea el problema consiste en realizar iteraciones de esta simulación para distintos tamaños de mundo en 3 distintas implementaciones, una secuencial hecha en CPU y dos paralelas usando las plataformas de CUDA y OpenCL para utilizar la GPU. Y para obtener un mayor entendimiento de estos algoritmos también se implementan versiones que poseen ineficiencias que en la teoría deberían ralentizar la ejecución para finalmente verificar como estas se traducen a las métricas finales, que para esta tarea corresponden al uso de *branching* y arreglos en 2D.

El propósito es evaluar el rendimiento para cada caso usando como medida la cantidad de celdas evaluadas por segundo, para así determinar la optimalidad de utilizar la GPU por sobre la CPU en problemas que son altamente paralelizables beneficiándose de gran manera de ello, además de también comparar la eficiencia entre las dos plataformas utilizadas para usar la GPU.

Adicionalmente, se espera que la implementación en CUDA sea más eficiente que la de OpenCL si se ejecutan ambas en el mismo dispositivo el cual posee una tarjeta gráfica dedicada de NVIDIA, al esta estar especializada para dicha arquitectura, junto con que la versión por defecto a implementar sea más eficiente que las que poseen código subóptimo a propósito.

2. Implementación

2.1. Secuencial en CPU

Se utilizó de base el código de ejemplo adjuntado en el enunciado para realizar la implementación inicial, el cual itera sobre cada celda, revisa la vecindad para determinar su nuevo estado, y actualiza los resultados tras haber iterado sobre todo, cabe destacar que la implementación considera un mundo que *loopea*, es decir, una celda de la última columna incluye de vecinos a los correspondientes de la primera columna, el análisis es análogo para las filas.

Un detalle es que se mapean los en 2 dimensiones a un arreglo de única dimensión y el chequeo de la vecindad se realiza sin hacer *branching*.

2.2. Paralela en CUDA

Nuevamente, se utilizó de base el código de la página que se incluyó, el cual realiza el mismo proceso que el punto anterior, con la única diferencia de que en vez de iterar sobre toda la grilla, se utilizan los *threads* de la GPU para determinar el valor de cada celda de forma paralela

2.3. Paralela en OpenCL

Para esto se tomó de inspiración el código realizado en CUDA para adaptarlo a OpenCL, de forma que se tenga el mismo comportamiento para ambos casos.

2.4. Variaciones en configuración

Como se mencionó anteriormente, se eligió el *branching* y arreglos en 2D como las variaciones a elección, de modo que para las 3 implementaciones se tuvo que adaptar el código base para manejar dicho cambio.

2.4.1. *Branching*

En este caso, se realizó un chequeo de la vecindad usando `if` por cada una de las 8 celdas que rodean la que se esté evaluando, adicionalmente se utilizó un `if` explícito para asignar el nuevo valor de la celda en vez de utilizar un operador ternario como se realiza en el código original.

2.4.2. Arreglos en 2D

Y luego, para esta variación se tuvo que adaptar el código para poder utilizar arreglos bidimensionales en cada caso, para la implementación secuencial fue directo de realizar usando un arreglo de arreglos, al igual que con CUDA.

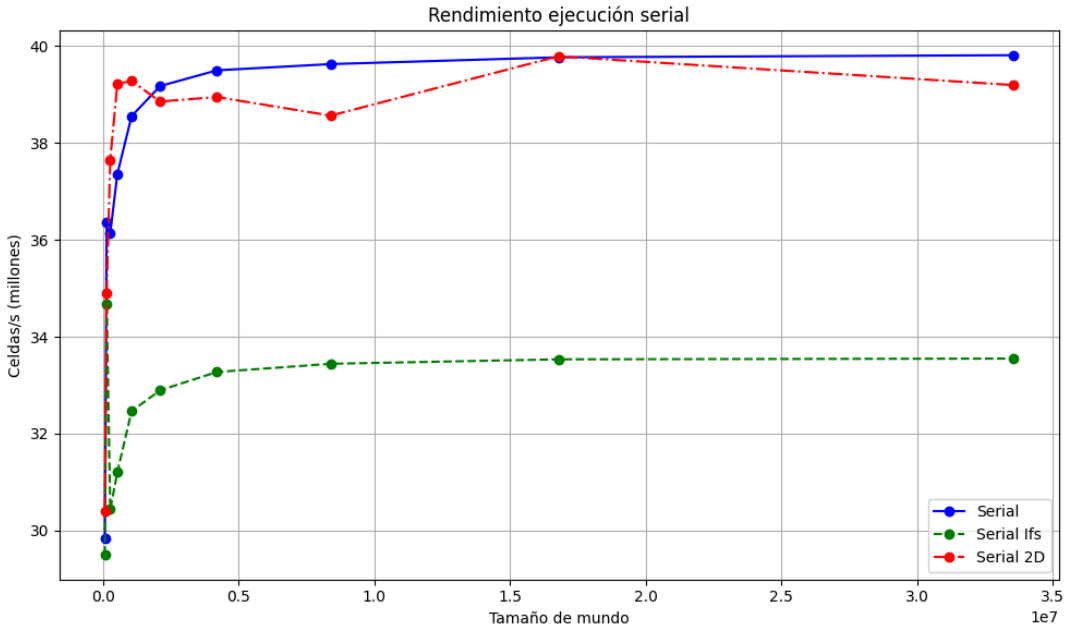
Pero para OpenCL fue más problemático, ya que este no soporta directamente el uso de arreglos con punteros a arreglos, lo cual sería lo normalmente utilizado para realizar esto, de modo que se utilizó una solución alternativa, siendo esta el uso explícito de coordenadas 2D al obtener los índices de cada *thread* para calcular el índice correspondiente pese a estar utilizando un arreglo de una dimensión.

3. Resultados

Todos los experimentos se hicieron usando tamaños de mundo en potencias de 2, tomando exponentes desde el 16 hasta el 30, exceptuando en el caso secuencial que solo se hizo hasta 26.

Para el caso serial se realizaron 5 ejecuciones distintas, cada una realizando 16 iteraciones del juego de la vida, en donde se eligió como tiempo final la mediana de estas 5 para tener resultados más consistentes, en cambio para las versiones paralelas se realiza un total de 15 ejecuciones distintas y tomando nuevamente la mediana.

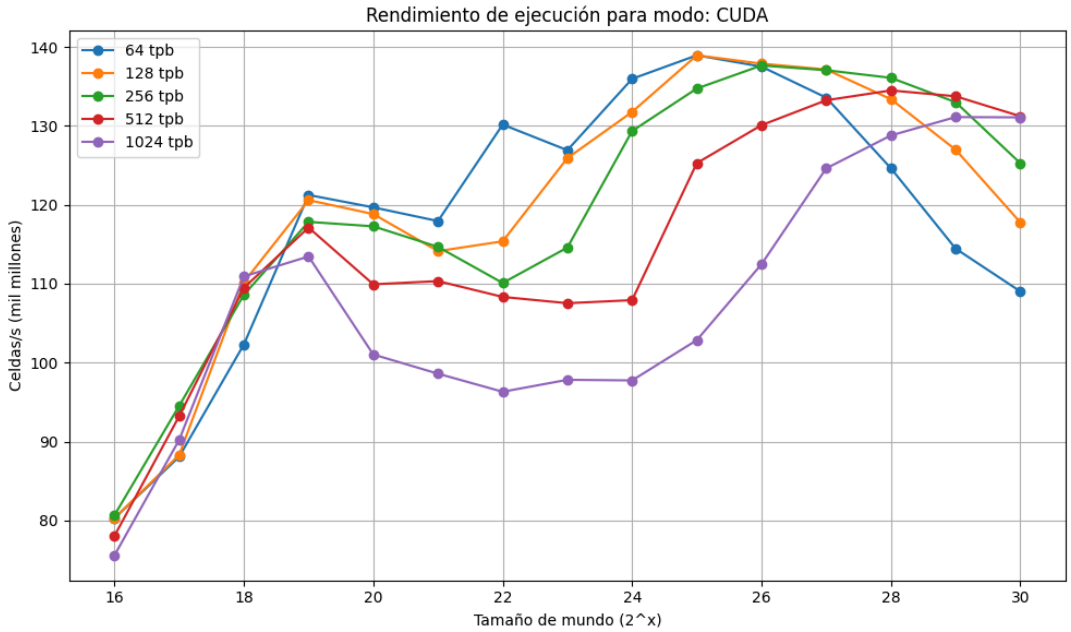
3.1. Serial en CPU



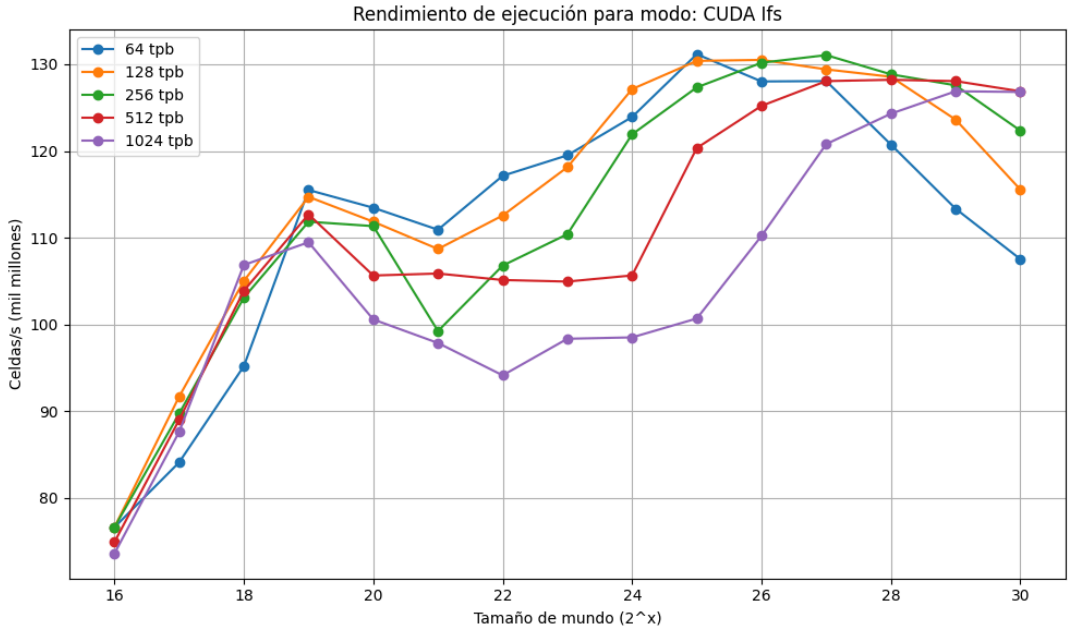
Ahora se expondrán los resultados para las implementaciones en GPU, incluyendo en ambas una curva por cada cantidad de *threads* por bloque (tpb) usados, que para este caso fueron: 64, 128, 256, 512 y 1024 (todos múltiplos de 32). Notar que las escalas verticales son distintas al caso secuencial.

3.2. CUDA

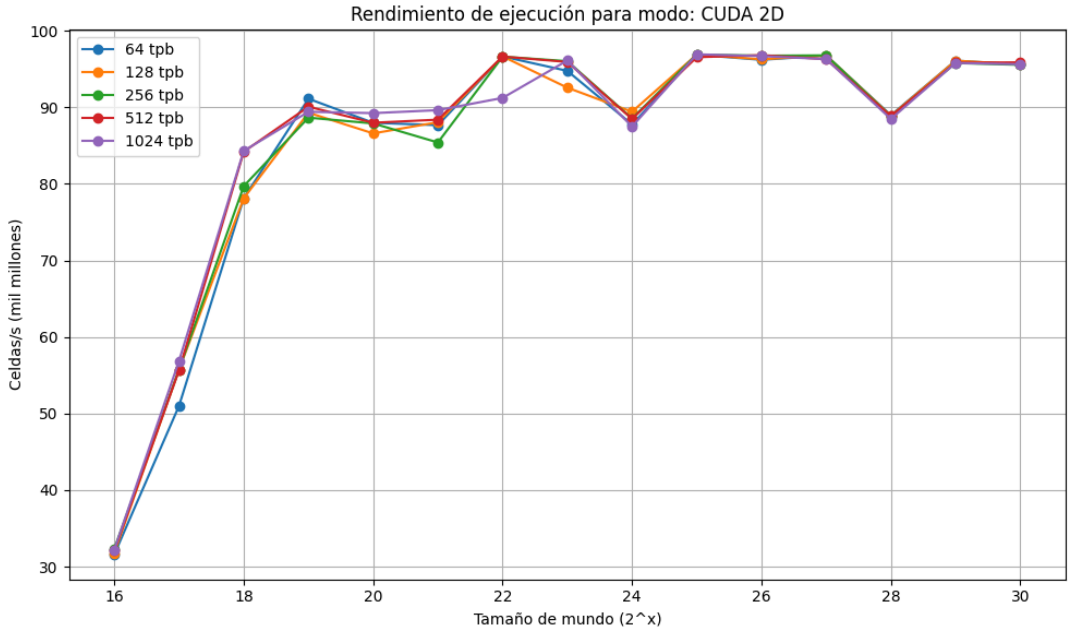
3.2.1. Sin variaciónes



3.2.2. *Branching*

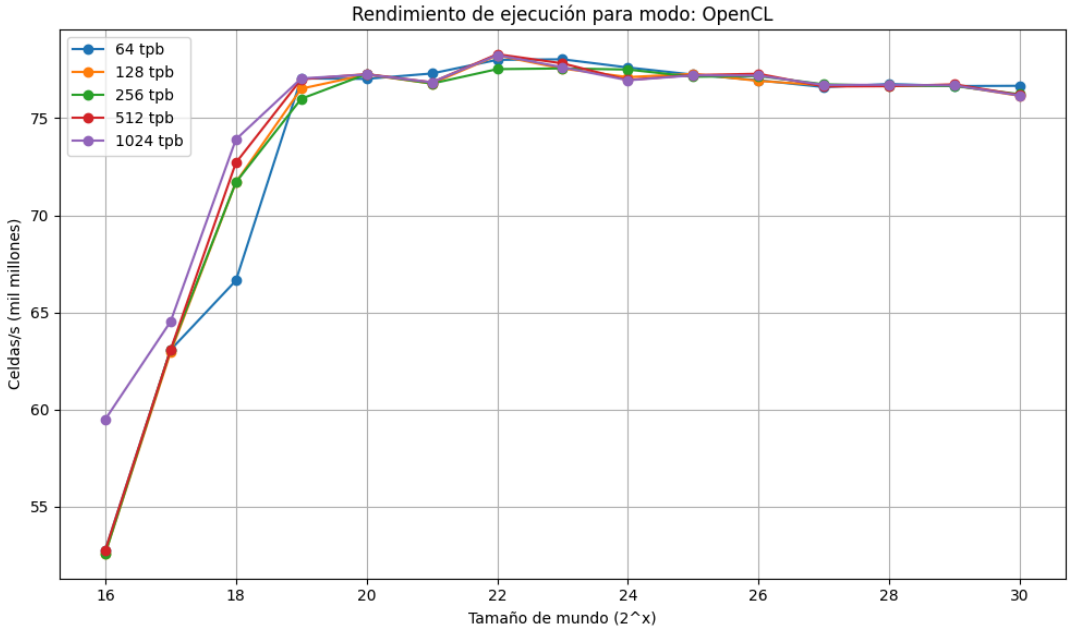


3.2.3. Arreglos 2D

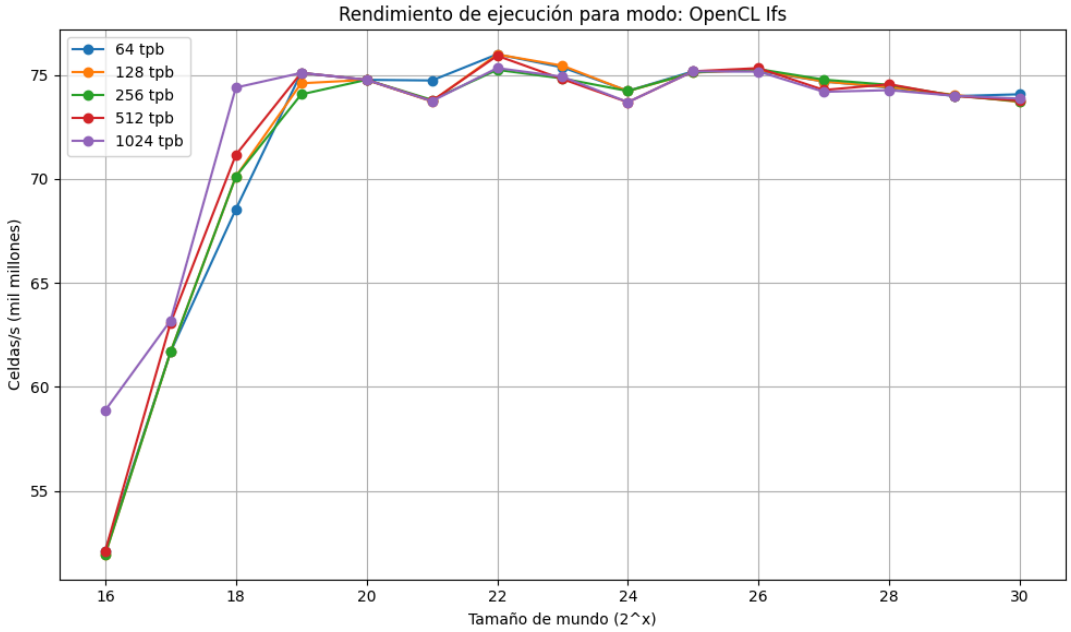


3.3. OpenCL

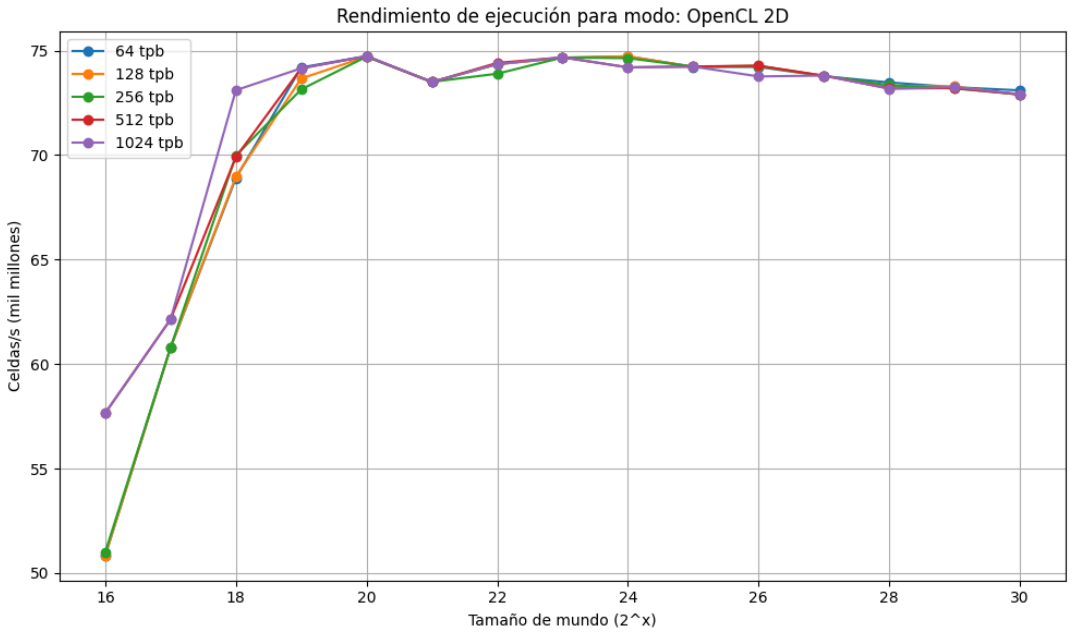
3.3.1. Sin variaciones



3.3.2. *Branching*

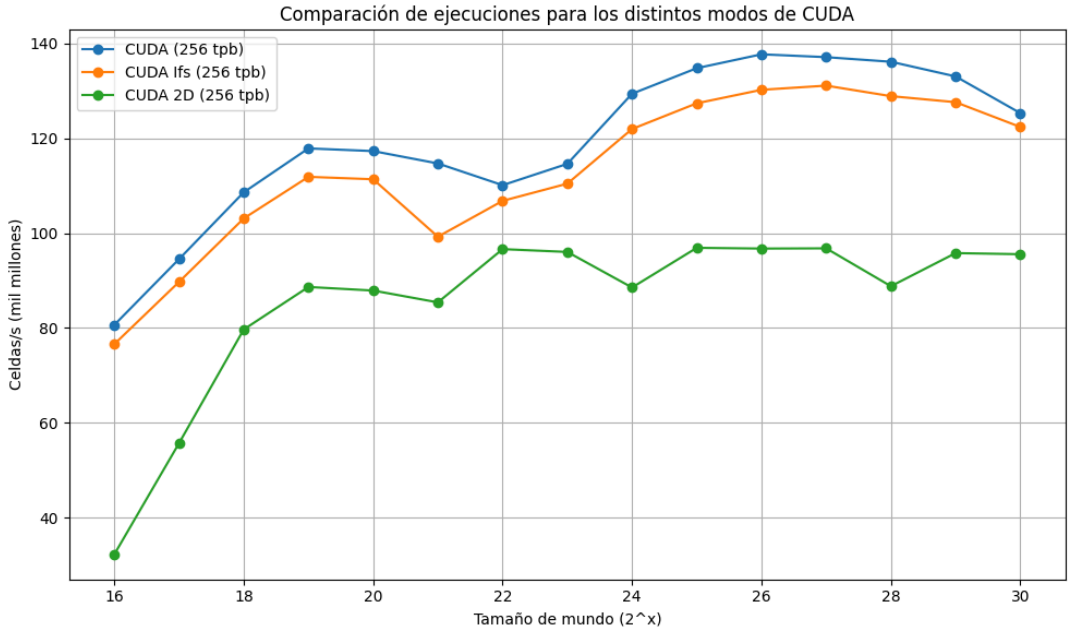


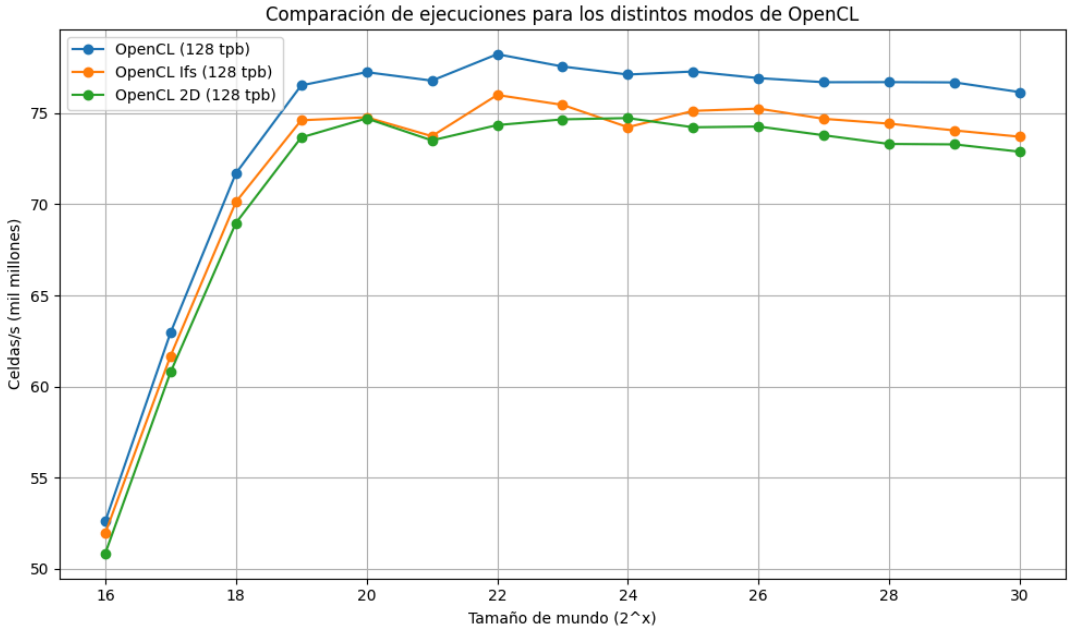
3.3.3. Arreglos 2D



3.4. Comparativa por tpb

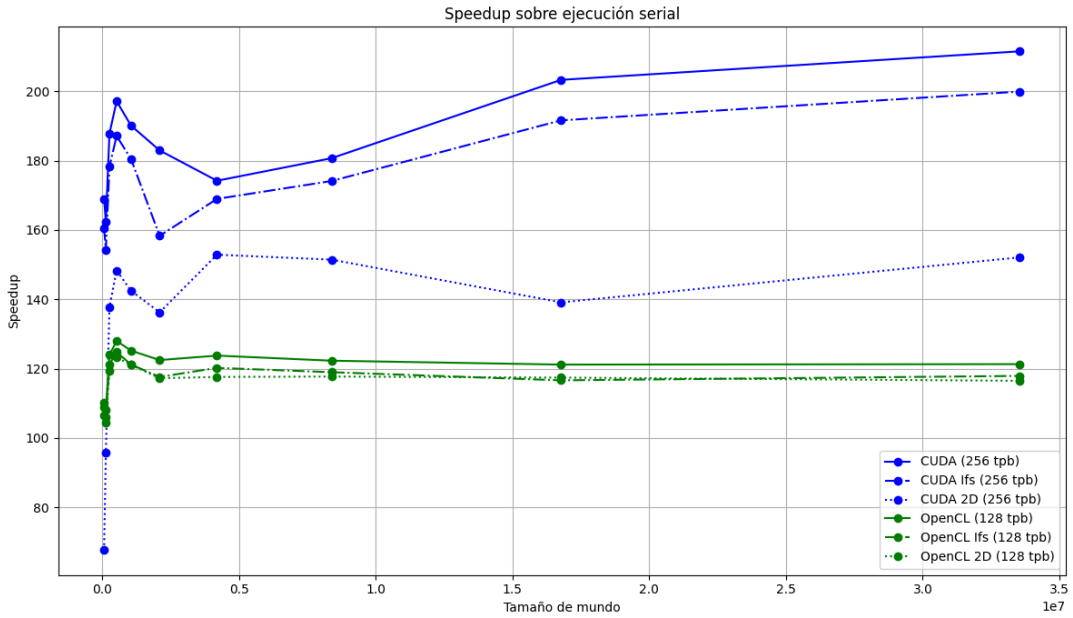
Finalmente, una comparativa de estos resultados eligiendo 256 tpb para CUDA y 128 tpb para OpenCL, ya que en general se manifiestan como las opciones más estables de entre las realizadas, para así tener una perspectiva en el rendimiento de cada plataforma utilizada junto a sus variaciones:




Comparación de ejecuciones para los distintos modos de OpenCL
OpenCL (128 tpb)
OpenCL lfs (128 tpb)
OpenCL 2D (128 tpb)
Celdas/s (mil millones)
Tamaño de mundo (2^x)
50
55
60
65
70
75
16
18
20
22
24
26
28
30

4. Análisis

A continuación se presenta el *speedup* logrado usando GPU al comparar con los valores obtenidos en las ejecuciones secuenciales de la CPU (se eligió 256 tpb para CUDA y 128 tpb para OpenCL con el fin de mantener la consistencia con los valores elegidos previamente):



De esto es posible ver que el *speedup* al usar la GPU es de al menos 2 ordenes de magnitud por sobre esta, notando que CUDA es más eficiente como se esperaba en un principio.

Y respecto a las métricas obtenidas en la sección de resultados, al incrementar el tamaño de grilla la cantidad de celdas evaluadas por segundo tiende a aumentar en todas las implementaciones, aunque es menos notable en la secuencial, pero este comportamiento llega hasta cierto punto y luego empieza a oscilar este valor alrededor de una constante según el caso.

Este comportamiento es mucho más notorio en los casos paralelos ya que mundos pequeños no aprovechan toda la capacidad de procesamiento de la que dispone la GPU.

Dados los resultados obtenidos para esta tarea, el *speedup* está presente en todos los tamaños de mundo, por lo que diremos que desde 2^{16}, que es el menor tamaño de mundo utilizado, es más conveniente utilizar la GPU, y para valores menores queda incierto según el estudio realizado en esta tarea.

5. Conclusiones

La hipótesis inicial fue que CUDA sería más eficiente que OpenCL en el uso de la GPU, argumentando que esto sería así ya que CUDA está especializado para el uso de GPUs de NVIDIA, mientras que OpenCL es de uso general, lo cual resultó ser correcto ya que efectivamente el *speedup* logrado con CUDA fue notablemente mayor que con OpenCL, especialmemte para tamaños de mundo grandes.

La otra suposición fue que las variaciones harían que la ejecución fuese menos eficiente ya que para GPU terminan siendo subóptimos dado el funcionamiento de estas, lo cual nuevamente terminó siendo verdadero, ya que como es posible evidenciar en los gráficos, la cantidad de celdas evaluadas por segundo es menor cuando se ejecutan estas variaciones en contraposición a cuando se tiene el código por defecto el cual no posee estas ineficiencias.

También cabe destacar que el *speedup* logrado demuestra que para tamaños desde 2^{16} es conveniente utilizar la implementación paralela en GPU, pero no es posible decir nada para tamaños menores ya que no se experimentó con estos, por lo que una forma de mejorar este estudio sería usando tamaños de mundo aún más pequeños que los realizados.

Y por último, nótese que habría que investigar si la implementación del *kernel* utilizado para simular arreglos 2D en OpenCL es realmente equivalente a algo como lo que se logró realizar con CUDA para poder afirmar que esos tiempos realmente sean válidos, ya que en esta tarea se asumió que esto fue así dado que los tiempos obtenidos parecían tener sentido si se comparaban con la teoría y también lo demostrado por CUDA.