Paralelizando el Método de Euler

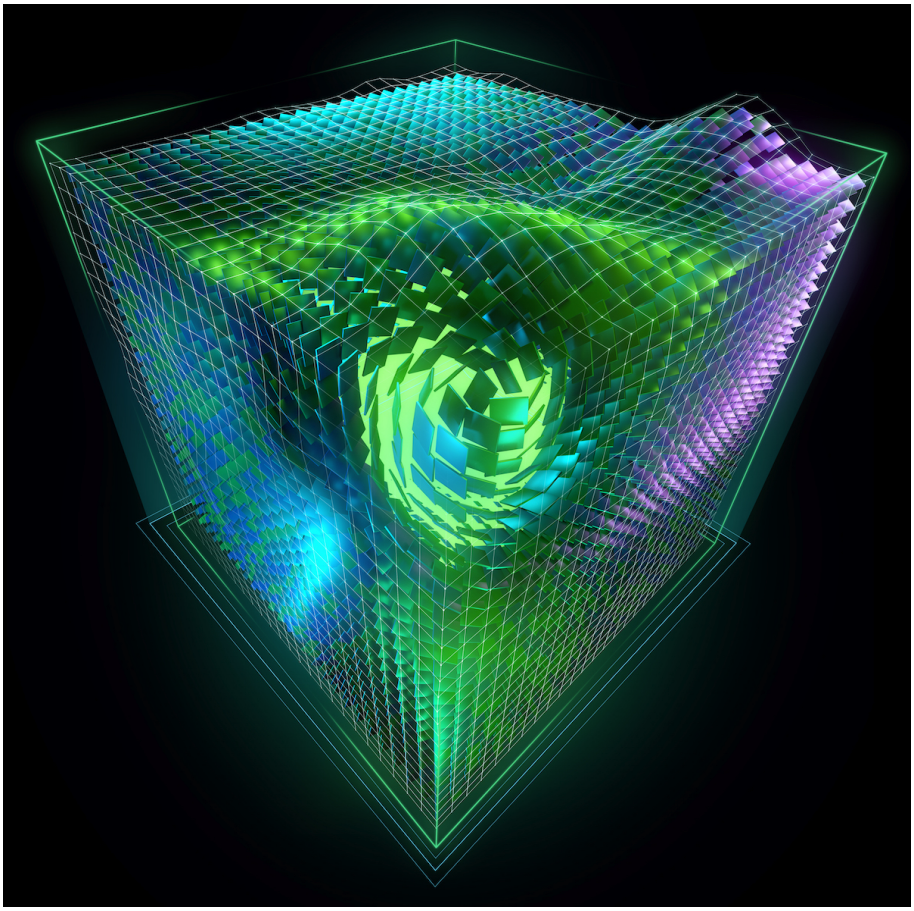
Laboratorio 1

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)

a)

```
Tiempo CPU para el valor delta t igual a 0.1 es: 0.046[ms]
Tiempo CPU para el valor delta t igual a 0.01 es: 0.039[ms]
Tiempo CPU para el valor delta t igual a 0.001 es: 0.394[ms]
Tiempo CPU para el valor delta t igual a 0.0001 es: 4.092[ms]
Tiempo CPU para el valor delta t igual a 1e-05 es: 39.702[ms]
Tiempo CPU para el valor delta t igual a 1e-06 es: 400.099[ms]
```

Imagen 1: Resultados CPU para Δt = $10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$ *y* 10^{-6}.

b) ¿Cuál de las dos implementaciones realizadas muestra un mejor desempeño? ¿Fue acaso conveniente utilizar GPU para este trabajo? Comente.

La implementación realizada mediante CPU es la que muestra mejor desempeño. Debido a que el problema tiene un enfoque secuencial, el paralelismo que provee el uso de GPU no mejora el rendimiento, ya que cada hebra va a tener que volver a realizar cálculos que ya fueron hechos o esperar a que otras hebras terminen, produciendo el problema de regularización. Además, debido a que los núcleos de la CPU poseen una mayor frecuencia que los de la GPU, la capacidad de cálculo de cada núcleo es mucho mayor, lo cual es beneficioso para este problema.

```
a) Tiempo CPU para el valor delta t igual a 0.1 es: 0.018[ms]
a) Tiempo CPU para el valor delta t igual a 0.01 es: 0.039[ms]
a) Tiempo CPU para el valor delta t igual a 0.001 es: 0.405[ms]
a) Tiempo CPU para el valor delta t igual a 0.0001 es: 4.154[ms]
----------------------
b) Tiempo GPU para el valor delta t igual a 0.1 es: 0.074592[ms]
b) Tiempo GPU para el valor delta t igual a 0.01 es: 0.669728[ms]
b) Tiempo GPU para el valor delta t igual a 0.001 es: 6.65536[ms]
b) Tiempo GPU para el valor delta t igual a 0.0001 es: 247.649[ms]
```

Imagen 2: Resultados CPU y GPU para Δt = $10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$.

c.1) ¿Cómo se comporta esta solución con respecto a las anteriores? Realice un gráfico o tabla en donde se muestren los tiempos de los 3 métodos implementados respecto al valor de *Δ*t que se utilizó para cada medición. Si opta por realizar un gráfico, obtendrá 3 curvas distintas (una por cada implementación), conteniendo el eje x los valores de *Δ*t, mientras que el eje y corresponderá a los tiempos medidos.

Considerando los resultados obtenidos para el nuevo método, podemos ver claramente que la solución híbrida es mucho mejor que la de solo GPU y se asemeja mucho más a la solución óptima de CPU, pero para valores mayores que 10^{-3} es mucho más lenta y para los valores más pequeños a este, las curvas de CPU y CPU-GPU son similares, pero notamos que la solución híbrida incrementa su tiempo de ejecución ligeramente en el valor10^{-6}, por lo que tendríamos que probar valores más pequeños para verificar si este aumento es una tendencia o simplemente es un outlier.

```
c) Tiempo CPU-GPU para el valor delta t igual a 0.1 es: 0.309[ms]
c) Tiempo CPU-GPU para el valor delta t igual a 0.01 es: 0.241[ms]
c) Tiempo CPU-GPU para el valor delta t igual a 0.001 es: 0.6[ms]
c) Tiempo CPU-GPU para el valor delta t igual a 0.0001 es: 4.346[ms]
c) Tiempo CPU-GPU para el valor delta t igual a 1e-05 es: 38.637[ms]
c) Tiempo CPU-GPU para el valor delta t igual a 1e-06 es: 408.078[ms]
```

Imagen 3: Resultados CPU-GPU para Δt = $10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$ *y* 10^{-6}.

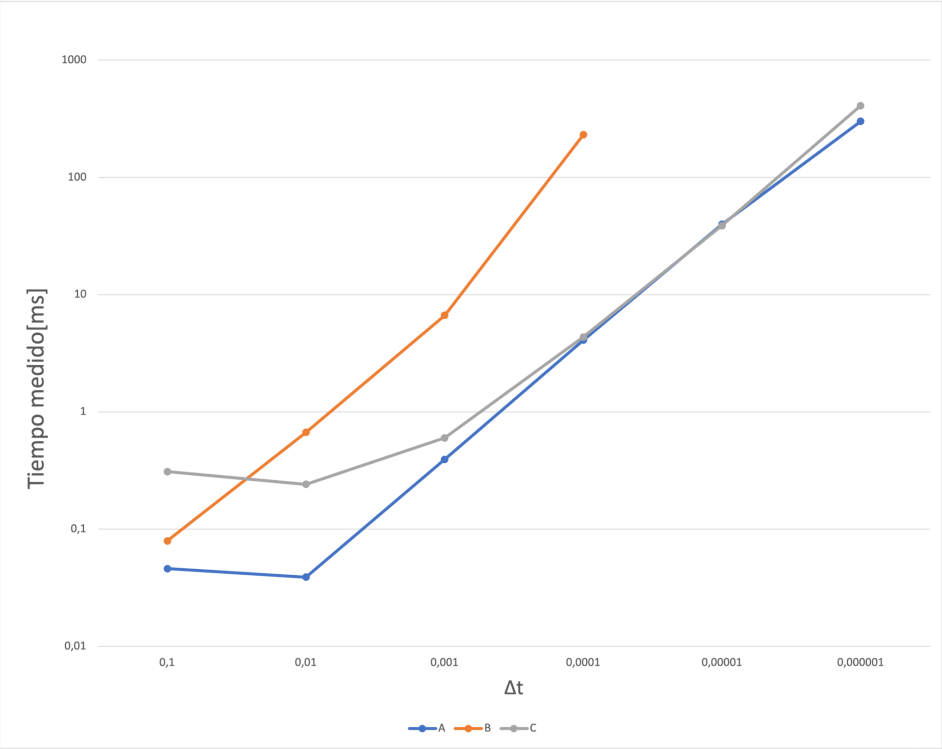


Gráfico 1: Comparación de resultados de A, B y C *(en escala logarítmica).*

c.2) Imagine el caso en que para la misma ecuación diferencial y tiempo t_i, se requiere resolver muchos problemas de valor inicial que se diferencian por su condición inicial y0. Argumente a favor de la solución que usted considera se demoraría menos en obtener resultados.

Como son varios problemas de valor inicial con valores iniciales distintos, se podría aprovechar el paralelismo que nos entrega la GPU por sobre la forma secuencial de la CPU, ya que, si bien la CPU es más rápida en términos de frecuencia, no nos permite resolver de manera rápida esta situación de varios problemas como lo haría la GPU debido a las hebras permiten resolver todos los problemas en paralelo y la CPU tendría que resolverlos uno a la vez.

2)

b) ¿Cuál de las dos soluciones es más rápida? ¿A qué se debe esto? Comente los resultados y realice un gráfico o tabla en donde se muestren los tiempos de ambas implementaciones respecto al valor $m$que se utilizó para cada medición. Si opta por realizar un gráfico, obtendrá 2 curvas distintas (una por cada implementación), conteniendo el eje $x$los valores de m,mientras que el eje y corresponderá a los tiempos medidos.

La solución de GPU es mucho más rápida en todas las iteraciones que la solución en CPU. Esto se debe a que son muchos distintos problemas, estos se pueden atacar de forma paralela, por lo que la GPU rinde mucho mejor al tener más núcleos capaces de paralelizar estos problemas.

```
a) Tiempo CPU para el valor de m igual a 10000 es: 33.423[ms]
----------------------
b) Tiempo GPU para el valor de m igual a 10000 es: 5.83261[ms]
----------------------
a) Tiempo CPU para el valor de m igual a 100000 es: 361.636[ms]
----------------------
b) Tiempo GPU para el valor de m igual a 100000 es: 5.89907[ms]
----------------------
a) Tiempo CPU para el valor de m igual a 1000000 es: 3411.89[ms]
----------------------
b) Tiempo GPU para el valor de m igual a 1000000 es: 28.0404[ms]
----------------------
a) Tiempo CPU para el valor de m igual a 10000000 es: 34463.3[ms]
----------------------
b) Tiempo GPU para el valor de m igual a 10000000 es: 355.482[ms]
----------------------
a) Tiempo CPU para el valor de m igual a 100000000 es: 343337[ms]
----------------------
b) Tiempo GPU para el valor de m igual a 100000000 es: 3435.75[ms]
```

Imagen 4: Resultados CPU y GPU para m = $10^4, 10^5, 10^6, 10^7, 10^8$.

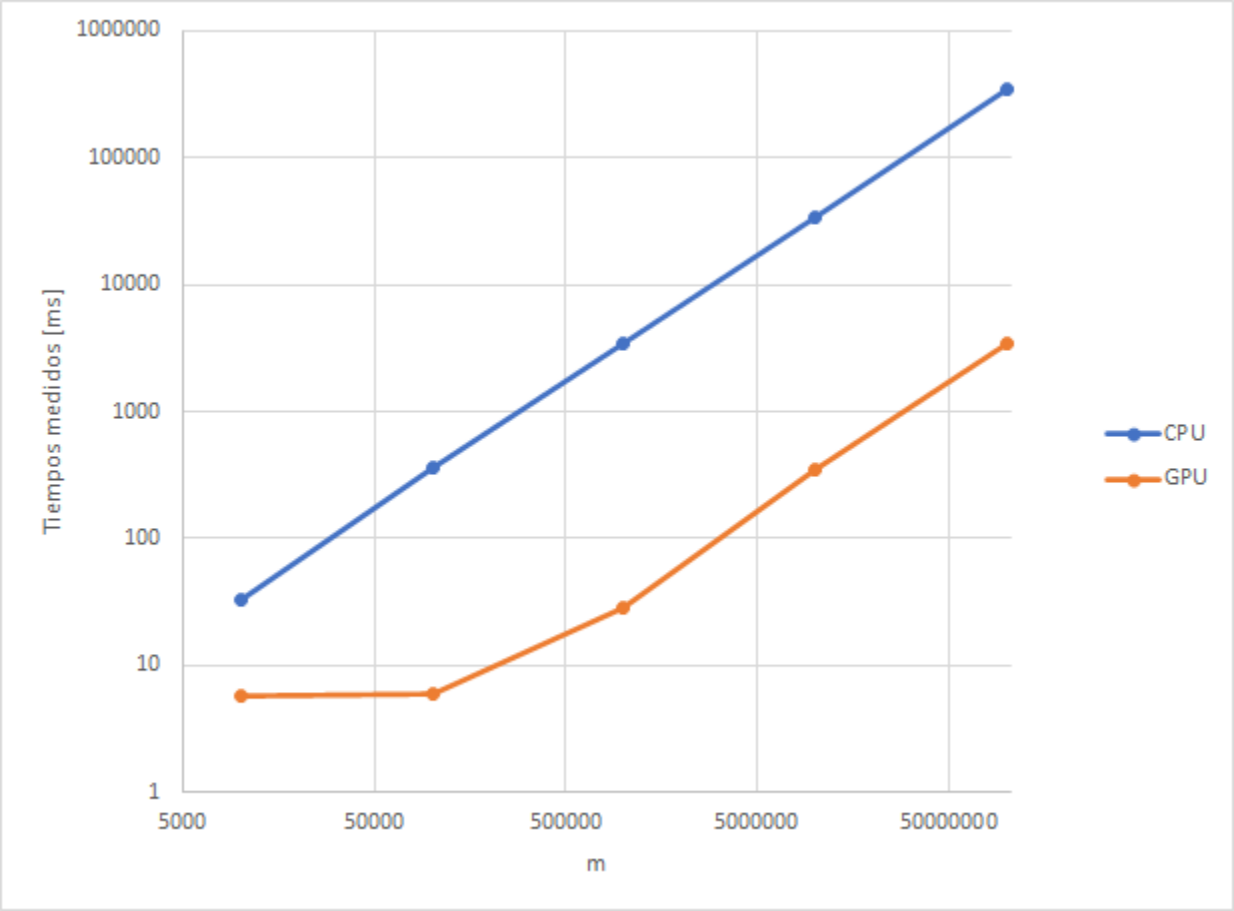


Gráfico 2: Resultados GPU y CPU vs m (en escala logarítmica).

c) ¿Con qué configuración se obtuvo el mejor resultado? Concluya sobre lo observado. Realice un gráfico o tabla en donde se muestren el tiempo medido respecto a los tamaños de bloque utilizados. Si opta por realizar un gráfico, obtendrá una curva, conteniendo el eje $x$los tamaños del bloque, mientras que el eje $y$corresponderá a los tiempos medidos.

Se obtuvo un mejor resultado con 128 hebras por bloque, esto es debido a que así puede tener más bloques activos en la SM y debido a la forma del ejercicio, ya que para arreglos de 2 dimensiones funciona mejor 128 o 256 hebras por bloque.

```
c) Tiempo GPU para el valor de m igual a 100000000 y bloque igual a 64 es: 3428.62[ms]
c) Tiempo GPU para el valor de m igual a 100000000 y bloque igual a 128 es: 3424.33[ms]
c) Tiempo GPU para el valor de m igual a 100000000 y bloque igual a 256 es: 3427.86[ms]
c) Tiempo GPU para el valor de m igual a 100000000 y bloque igual a 512 es: 3428.7[ms]
```

Imagen 5: Resultados GPU con 64, 128, 256 y 512 hebras por bloque.

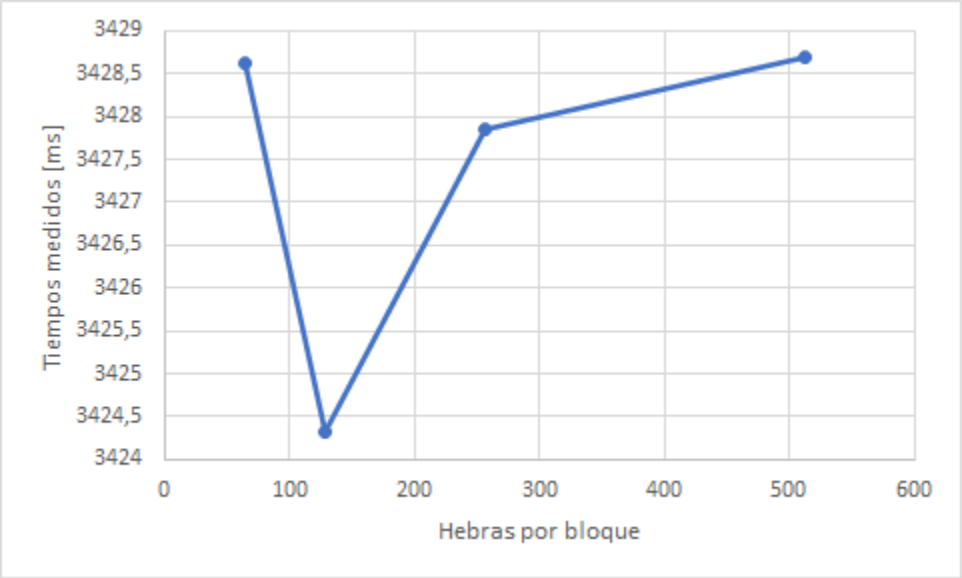


Gráfico 3: Resultados tiempos GPU vs cantidad de hebras por bloque.

Conclusión

Al realizar la tarea pudimos concluir que, dependiendo de la tarea que se le exija al computador, la CPU o GPU funcionará mejor. Por ejemplo, los procesos secuenciales son mejores para la CPU debido a que las hebras se bloquean y esperan a las demás, por lo que es mejor tener pocas hebras rápidas, que muchas hebras lentas. Por otro lado, los problemas que no necesitan de un resultado previo, pueden realizarse de forma paralela, por lo que la GPU funciona mejor debido a la gran cantidad de núcleos capaces de realizar tareas de forma simultánea.

Durante el laboratorio (en la pregunta 2c) realizamos el supuesto de que siempre funcionará mejor con 128 hebras por bloque, lo cual ocurría la mayoría de las veces, siendo también posible que funcionara mejor con 256 hebras por bloque (aunque con menos probabilidad).