Laura Llamas López - alumno09

Informe Laboratorio 6. GPUs con Python.

Este laboratorio ha resultado particularmente valioso para comprender en profundidad qué son las GPUs y su papel en la aceleración de cómputo científico. Aunque el concepto de GPU es omnipresente en el ámbito informático actual, especialmente con el auge de la inteligencia artificial, sin una explicación detallada puede resultar abstracto y difícil de entender. La práctica ha permitido no solo comprender su función de acelerar tareas computacionalmente intensivas, sino también conocer su evolución histórica desde su propósito original en renderizado gráfico hasta convertirse en un componente fundamental para cálculo científico y *machine learning*.

La lectura previa de la teoría antes de abordar el laboratorio ha sido esencial para evitar errores comunes. En particular, comprender que las operaciones en GPU son asíncronas y que por tanto no se puede usar simplemente `%timeit` o `time.time()` sin sincronización ha sido crucial. El concepto de `cuda.synchronize()` y la necesidad de esperar a que la GPU termine sus cálculos antes de medir tiempos es algo que solo se entiende realmente al trabajar directamente con estos dispositivos. En este sentido, la función `benchmark` de CuPy ha resultado muy práctica al encapsular toda esta complejidad.

El material de clase ha sido un recurso valioso para complementar el laboratorio. Las funciones de timing para Numba (`timing_numba_GPU` y `timing_numba_GPU2`) proporcionadas han permitido comparar diferentes metodologías de medición y verificar la consistencia de los resultados obtenidos con `benchmark`.

El trabajo con CuPy ha resultado especialmente intuitivo al ser prácticamente un equivalente de NumPy para GPU. Numba con el decorador `@vectorize(target='cuda')` ha presentado un enfoque distinto que, si bien potente, requiere adaptar el código a sus limitaciones. El ejercicio del cálculo de π ha ilustrado claramente esta restricción: al no poder realizar reducciones como `sum()` dentro de la ufunc GPU, parte del procesamiento debe volver a CPU. Esta limitación arquitectónica explica por qué en algunos casos CuPy puede ofrecer mejor rendimiento que Numba para ciertas operaciones, dado que CuPy mantiene todo el cálculo en GPU. La introducción a Pytorch en la actividad extra ha sido particularmente interesante, presentando el concepto de tensores.

La práctica también ha evidenciado de forma muy clara el impacto del *overhead* de transferencia de datos entre CPU y GPU. Los resultados consistentemente muestran que las copias de memoria pueden consumir la mayor parte del tiempo de ejecución, haciendo que en algunos casos el beneficio de usar GPU se reduzca drásticamente o incluso desaparezca. Esta lección sobre la importancia de minimizar transferencias y mantener los datos en GPU el mayor tiempo posible es fundamental para el diseño eficiente de aplicaciones GPU.

En cuanto a aspectos mejorables, habría sido útil incluir una discusión más detallada sobre cuándo merece la pena usar GPU versus CPU. Asimismo, dado que Pytorch no era compatible con la GPU de nikola y solo funcionaba en bohr, habría sido conveniente especificar esto más claramente desde el principio del enunciado para evitar intentos de ejecución en el entorno incorrecto.

A pesar de estos detalles menores, el laboratorio cumple efectivamente su objetivo de introducir el uso de GPUs desde Python. Las competencias adquiridas sobre CuPy, Numba y Pytorch son directamente aplicables a problemas reales de bioinformática y ciencia de datos donde la aceleración GPU puede marcar la diferencia entre análisis viables e inviables.