HPC TOOLS AI: DISTRIBUTED

Unai Iborra Albeniz
iborra.unai@gmail.com

Gonzalo Silvalde Blanco
gonzalo.silvalde@udc.es

Resumen

En este documento se detalla la segunda práctica de **HPC TOOLS AI** donde hemos mejorado la implementación secuencial básica de **BERT** a un entorno distribuido utilizando PyTorch Lightning. Se analizan las estrategias de paralelismo de datos (DDP), la optimización de la carga de datos y los resultados obtenidos tras un breve entrenamiento.

1. Implementación Distribuida con PyTorch Lightning

Para superar las limitaciones de rendimiento inherentes a la ejecución en una sola GPU y mejorar la escalabilidad del entrenamiento, hemos refactorizado la implementación original utilizando el framework **PyTorch Lightning**. Esta migración ha permitido abstraer la complejidad del bucle de entrenamiento manual y facilitar la ejecución distribuida en 2 nodos (y por tanto 4 GPUs) sin alterar la lógica del modelo.

La nueva arquitectura de software se divide en dos componentes desacoplados: `SQuADDataModule` para la gestión del pipeline de datos y `BERTSQuADModule` para la lógica del modelo.

1.1. Gestión de Datos y Paralelismo

La clase `SQuADDataModule` centraliza la carga, el preprocesamiento y la creación de *dataloaders*. A diferencia de la implementación secuencial anterior, este módulo introduce mecanismos de sincronización explícita (`strategy.barrier()`) para garantizar que, en un entorno multiproceso, solo el proceso principal (rango global cero) realice la descarga y el preprocesamiento costoso, mientras que los demás procesos esperan y cargan posteriormente los datos procesados desde la caché.

Adicionalmente, se ha optimizado la transferencia de datos entre la CPU y la GPU configurando los *dataloaders* con `num_workers=4` y `pin_memory=True`. Esto permite la carga asíncrona de datos en la memoria paginada y reducir los cuellos de botella.

1.2. DistributedDataParallel (DDP) y Precisión Mixta

La mejora más significativa en esta iteración es la adopción de la estrategia **DDP** (*Distributed Data Parallel*). El modelo se replica en cada GPU disponible en los 2 nodos. Durante el *backward pass*, los gradientes se calculan localmente y se sincronizan a través de todos los dispositivos antes de la actualización de los pesos, garantizando la consistencia del modelo.

Para optimizar aún más el uso de memoria y la velocidad de cómputo, hemos activado el entrenamiento con precisión mixta. Esta técnica permite realizar la mayoría de las operaciones matriciales utilizando tipos de punto flotante de 16 bits (FP16) en lugar de los 32 bits estándar (FP32), manteniendo FP32 solo para operaciones sensibles como la acumulación de gradientes.

1.2.1. Otras técnicas

Aunque intentamos aplicar otras de las técnicas propuestas como puede ser **ZERO** o **FSDP**, la falta de tiempo y la cantidad de errores que aparecieron durante las ejecuciones, nos llevo a utilizar simplemente **DDP**.

1.3. Profiling

Para analizar el rendimiento de la implementación distribuida se ha utilizado el `AdvancedProfiler` de PyTorch Lightning. Esta herramienta proporciona una tabla en formato texto con los tiempos obtenidos durante la ejecución distribuida. Los resultados completos pueden consultarse en el archivo `results/fit-profile_2gpu.txt-X.txt`.

Los resultados muestran que la fase de `setup` consume una gran parte del tiempo total de ejecución, al igual que el `mapping`. La influencia global de estas funciones podría reducirse aumentando el número de épocas, ya que se ejecutan antes del entrenamiento real del modelo y no en cada época. Respecto a la ejecución del entrenamiento del modelo, no se han encontrado problemas de rendimiento ni cuellos de botella significativos en el análisis de *profiling*.

2. Resultados y Conclusiones

Los experimentos se realizaron utilizando 2 nodos, sumando un total de 4 GPUs Nvidia A100. Se comparan a continuación con el **baseline** secuencial obtenido en la práctica anterior (mismo número de epochs).

Tabla 1: Comparativa de Rendimiento (5 Epochs)

Métrica	**Baseline (1 GPU)**	**DDP (4 GPUs)**
Tiempo Total	$\approx$ 14.0 min	**1.98 min**
Precision	FP32	FP16
Speedup	1x	$\approx$ **7.07x**

Como se observa en la Tabla 1, el tiempo se reduce de 14 minutos a menos de 2 minutos.

2.1. Análisis del Speedup

Se ha obtenido un *speedup* aproximado de 7x. Este valor supera el máximo teórico de 4x (correspondiente al número de GPUs). Atribuimos esta mejora adicional a la combinación de tres factores: el uso de **Mixed Precision** en la versión distribuida (que acelera las operaciones tensoriales), el aumento del *batch size* efectivo y la eficiencia de los dataloaders optimizados en Lightning.