Assignment 2: Understanding LLM Performance

In this assignment, we will perform deep performance profiling for LLM and understand the motivation and benefits of FlashAttention.
You will write a report that contains all of your plots as well as your explanations on the profiling results.

Preliminary

For this homework, we will run Python scripts that implement Llama's Decode layer and commonly used operators in LLMs. We will measure their performance on AMD's GPU.
Our code is implemented based on PyTorch and the HuggingFace Transformer library.
After you get access to the AMD cluster, request a job with a node equipped with `MI2104x` GPUs.
You can execute the scripts by submitting a `sbatch` job or running it in an `interactive` session with a limited job time (less than 30min).

Notice: Please do *NOT* run any jobs on the login node of the AMD cluster.

Before executing any of these scripts, you need to load the `PyTorch` module via the command below in the compute node environment of the AMD cluster.

```
module load pytorch
```

To know what modules are currently loaded and what modules are available, you can run `module list` and `module avail`.

An example sbatch script `hw2.sbatch` is also provided. Run the following command to launch a job.

```
sbatch hw2.sbatch
```

You can also run the script by using the `srun` command.

```
module load pytorch
srun -N 1 -n 1 -t 00:30:00 -p devel -o output.%J.log --exact python bench_llama.py
```

Please also check out the AMD cluster usage guideline for detailed guidance.

Part-I: Profile Llama Decode Layer (30pt)

We provide the `bench_llama.py` script, which closely implements the Decode layer block of the Llama model based on the Transformer library.
The script `bench_llama.py` profiles the latency of the Decode layer and breaks down the profiling into three parts: Self-Attention, MLPs, and misc.

Note: For the example commands below, **we expect you to use `srun` or `sbatch` to run it**. The output will be in the output file.

1.(a) Prefill stage (10pt)

In this part, we will profile the performance of the Prefill stage. We use random initialized tensors
To get an example profiling with `batch 1` and `sequence_length 1024` for `Prefill`, please run.

```
python bench_llama.py --stage prefill --batch 1 --seq-len 1024
```

You shall get the output below.

```
Prefill Stage Time Results (ms)
Input size: Batch=1, Seq_len=1024, Hidden_dim=4096
Self-Attn,     MLPs,     Misc,     Total
    1.967,   13.178,    0.484,    15.628
```

- **Q1 (5pt)** - Set `batch = 1` and benchmark the `Prefill` stage latency with `seq_len=[256,512,1024,2048,4096,8192,12288]`. Plot the latency of Self-attention and MLPs and explain the results.
- **Q2 (5pt)** - Set `seq_len = 1024` and sweep the batch size from `[1,2,4,8,16,32,64]`. Plot the latency of Self-attention and MLPs again and explain the results.

1.(b) Decode stage (20pt)

In this part, we will profile the performance of the Decode stage. You will need to implement the missing section in the `bench_decode()` function before you can run the profiling.

```
# Implement the part that is marked as below
# All you need to do is to declare random PyTorch tensors that match the shapes of
########## Solution Block ##########
########## End Solution Block ##########
```

After completing the `bench_decode` function, you can profile the `Decode` stage with the following command:

```
python bench_llama.py --stage decode --batch 1 --seq-len 1024
```

- **Q3 (5pt)** - Complete the `bench_decode()` function.
- **Q4 (5pt)** - Redo **Q1** in 1.(a)
- **Q5 (5pt)** - Redo **Q2** in 1.(b)
- **Q6 (5pt)** - What is the dominant operator of the `Prefill` and `Decode` stages when the sequence length is long?

Part-II: Profile GEMM and Self-attention (70pt)

In Part-II, we will dive deeper into understanding the performance of the dominant operators, GEMM and Self-Attention, in LLMs.
We provide a script, `bench_llm_ops.py`, to benchmark the latency of these operators.
The default `number_of_heads`, `embeded_dim_per_head` and `hidden_dim` in the script are set to `32, 128, 4096`, which match Llama-7B's configuration.
We will use the throughput v.s. arithmetic intensity plot to help understand the performance.

2.(a) GEMM (20pt)

To benchmark the latency of GEMM operation for certain batch size and sequence length, please run the following command.

```
python bench_llm_ops.py --bench gemm  --batch 16 --seq-len 128
```

You shall get the benchmarking results as below.

```
Benchmarking for GEMM in ms
Input size: Batch=16, Seq_len=128, Hidden_dim=4096
   Total
   0.534
```

- **Q7 (10pt)** - Set `hidden_dim=4096`, benchmark GEMM's latency for the combinations of `batch=[1,4,8,16,32,64]` and `seq_len=[1,64,128,256,512]`. Calculate the compute throughput (FLOPs/s) and arithmetic intensity (FLOPs/Bytes) for each data point and plot them in a figure **(5pt)**, where X-axis is the arithmetic intensity and Y-axis is the compute throughput. You can use different colors for dots of different sequence lengths. Provide a writeup on the profiling results **(5pt)**.
- **Q8 (10pt)** - Set `hidden_dim=8192`, re-do **Q7**. Notice `hidden_dim=8192` is the configuration used in Llama-70B model.

2.(b) Attention and Fused attention - Prefill (20pt)

To benchmark the latency of the self-attention operation for the `Prefill` stage given a certain batch size and sequence length, run the following command.

```
python bench_llm_ops.py --bench attn --stage prefill  --batch 16 --seq-len 256
```

You shall get the benchmarking results as below.

```
Benchmarking for Attention prefill stage in ms
Input size: Batch=16, Seq_len=256, Num_heads=32, Embed_dim=128
   Q_K^T,  Masking,  Softmax,   Attn_V,     Total
   0.294,    0.210,    0.453,    0.534,     1.491
```

To benchmark the latency of the fused self-attention implementation provided in PyTorch, put `--fused` in the command.

```
python bench_llm_ops.py --bench attn --stage prefill  --batch 16 --seq-len 256 --f
```

You shall get the benchmarking results as below.

```
Benchmarking for Attention prefill stage in ms
Input size: Batch=16, Seq_len=256, Num_heads=32, Embed_dim=128
    Total
    0.381
```

- **Q9 (10pt)** - Benchmark the latency of the regular attention for `seq_len=[256,512,1024,2048,4096,8192,12288,16384]` and `batch=[1,4,8,16,32]`. Ignore the cases when the GPU is out of memory. Calculate the plot of the compute throughput vs. arithmetic intensity figure. You can use different colors for dots of different batch sizes. Explain the results in your writeup.
- **Q10 (10pt)** - Benchmark the latency of the fused attention and calculate the speedup against the regular attention. Plot the speedups in a line chart. You can use different colors for lines of different batches. Explain the results in your writeup.
- **Bonus (5pt)** - Given the profiling results, provide some ideas that can further accelerate the attention in `Prefill` stage and explain why.

2.© Attention and Fused attention - Decode (30pt)

In this part, we will profile the attention performance for the Decode stage. You will need to implement the missing section in the `bench_attn_decode()` function in `bench_llm_ops.py`. The missing section is marked as in **1.(b)**.

After completing the `bench_attn_decode()` function, you can run the profiling with

```
python bench_llm_ops.py --bench attn --stage decode  --batch 16 --seq-len 256
```

You shall get the benchmarking results as below.

```
Benchmarking for Attention decode stage in ms
Input size: Batch=16, Seq_len=256, Num_heads=32, Embed_dim=128
   Q_K^T,  Masking,  Softmax,   Attn_V,     Total
   0.172,    0.001,    0.016,    0.127,     0.316
```

- **Q11 (10pt)** - Implement `bench_attn_decode()` function. Implement it in a way that the KV-Cache optimization is used.
- **Q12 (5pt)** - Set `batch=16`. Benchmark the latency of the regular attention for `seq_len=[256,512,1024,2048,4096,8192,16384,32768]`. Calculate the plot of the compute throughput vs. arithmetic intensity figure. Explain the results in your writeup.
- **Q13 (5pt)** - Set `seq_len=2048`. Benchmark the latency of the regular attention for `batch=[16,32,64,128,256,512,1024]`. Calculate the plot of the compute throughput vs. arithmetic intensity figure. Explain the results in your writeup.
- **Q14 (10pt)** - Benchmark latency of the fused attention for the cases we tested in **Q11 and Q12**. Compare the speed of fused attention and regular attention. Explain the results in your writeup.
- **Bonus (5pt)** - Given the profiling results, provide some ideas that can further accelerate the attention in `Decode` stage and explain why.

Submission

Put your report in PDF format. Put the report and code under a single folder. Compress the folder and name your compressed file as `firstname_StudentID.tar.gz`. Submit it to Canvas.