E0 294: Systems for Machine Learning, Jan'25
Indian Institute of Science, Bangalore
Assignment #4
Deadline: 25th March, 11:59 PM

(The numbers in the braces denote the points)

Please write the solution in latex. Put all your codes and the PDF into a zip file and submit. The submitted code should be properly commented and the name of the variables should be intuitive. The code should be implemented by yourself. Marks will be deducted due to codes without comments, with non-intuitive variable names, with high similarity.

Problem 1 : Let's do some math (25 points)

KV cache, batch size, and arithmetic intensity calculation: Write a script (in any language of your choice), that given model, hardware, and input specs enumerated below, computes the size of the KV cache per request and the maximum permissible batch size on the given hardware.

The following are the inputs to the program :

Model Specification
Number of Layers
Number of Query heads
Number of KV heads
Embedding Dimension (h1)
Inner Dimension (h2)
Vocab Size

Device Specification
GPU Memory

Input Specification
Context Length (s)

Quantization
Weights bytes (eg., 2 if 16bit)
KV cache bytes

Consider following parameters

The default values populated are for Llama3-8B.

Number of Layers	32
Number of Query heads	32
Number of KV heads	8
Embedding Dimension (h1)	4096
Inner Dimension (h2)	14336
Vocab Size	128256
GPU Memory	80 GB

(a) Find the maximum batch size that can fit on a GPU given the above specifications. Please output a table in the below format and show calculation for how you find KV cache per request. (10)

Total model parameters	
Total parameter memory (GB)	
KV cache per request (GB)	
Max batch size possible	

(b) Repeat the same exercise for a given Tensor Parallelism (TP) dimension 'X'. Assume the model and KV cache memory requirements are divided equally among X GPUs, and recalculate the maximum batch size. (5)

(c) Write down the detailed formula for total flops and total memory access(bytes) to calculate arithmetic intensity(flops/memory_bytes) for prefill and decode attention function. Assume the following attention implementation. Head dimension is d and context length is N.

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Write your calculations, specifically how you arrive at the total flops and memory access for both prefill and decode attention in terms of N and d (7)

(d) Calculate arithmetic intensity using the formula above, given N=1024 and d=128. Plot the arithmetic intensity as context length (N) varies (for prefill, batch size 1) and as batch size varies (for decode). Crisply explain your observations and reason. (3)

Problem 2 : Transformer implementation with KV caching (25 points)

Implement a simple 2 layer decoder-only transformer model in Python, using the provided skeleton code.

(a) Implement all the TODO sections in the provided code to create a functioning transformer model with the ability to autoregressively generate 'n' tokens. (5)

(b) Correct implementation of the model with logits generated after the initial prompt processing matching the expected values for a given model weight file.
Run the file with mode --evaluate (5)

(c) Implement KV caching to cache the generated KV from previous iterations in every subsequent iter instead of naively processing the entire sequence in each step. Validate that the logits with and without KV caching match
Run the file with mode –kv_evaluate (7)

(d) Vary the sequence length from [10, 50, 100, 500, 1000] and compare the performance(latency to generate ‘n’ tokens given a prompt) with and without kv caching of prior tokens. Plot a graph that shows latency with and without KV as a function of sequence length.
Run the file with mode –benchmark (8)
(Partial : Half the points if you are not able to implement the generate method, but can demonstrate the impact of KV cache with just prompt processing)