Coding Test: Parameter Offload

Introduction

To reduce GPU memory usage, a common technique in deep learning is "parameter offload," which saves temporarily unused parameters into CPU memory. In this coding test, your task is to implement parameter offload for LLMs. This test only requires you to have basic knowledge of deep learning, Python, and PyTorch. It is designed to examine your coding ability and is not directly related to the ongoing research topic. You will need access to a single GPU to complete this test.

IMPORTANT NOTE: You are not allowed to use external libraries other than `transformers` and PyTorch in your implementation. However, feel free to use any available tools to implement this test, including ChatGPT, GitHub, etc. You can also copy open-source code into your implementation as needed.

Instructions

1. Implement an inference example with the LLaMA model on GPU.
   Hints:
   - "Inference" refers to a single forward pass over a batch of tokens.
   - You can find a public implementation of the LLaMA model in the `transformers` library.
   - You don't need to load the pre-trained LLaMA model, as it may not fit into your GPU memory. Instead, randomly initialize a customized LLaMA model that can fit in your GPU by adjusting the configuration in the model files.
   - The precision of parameters could be `torch.float16` if your GPU supports it; otherwise, use `torch.float32`.
   - Input data could be random numbers, and the batch size should be 4096 tokens.
2. Change the device of the previous example to CPU.
3. Load each layer to GPU only when it is required during inference.
   Hints:
   - "Layer" refers to a `LlamaDecoderLayer` or a transformer block. You only need to consider offloading at this level.
   - Implement a customized `forward` function for the LLaMA model. In this function, the forward call of each layer is associated with two additional operations: (1) load it to GPU before calling `layer.forward`, and (2) store it back to CPU after calling `layer.forward`.
   - You may find that the time for a single inference significantly increases due to these operations. We will address this issue in the subsequent steps.
4. Utilize CUDA streams to overlap parameter loading/storing and computation (i.e., `layer.forward`).
   Hints:
   - Ideally, prefetch a layer from the CPU and overlap it with the computation of previous layers. Thus, when this layer is required for computation, it resides exactly on the GPU, eliminating the need to wait for expensive memory copying.
   - You may find the `non_blocking` argument in `torch.Tensor.copy_` useful.
   - Examples of using CUDA streams:
     - `https://developer.nvidia.com/blog/how-overlap-data-transfers-cuda-cc/`
     - `https://zhuanlan.zhihu.com/p/66145913`
   - You can use the PyTorch profiler to monitor the trace of CUDA kernel execution. With an overlapped implementation, you should observe two independent CUDA streams with concurrent kernel calls in the produced trace.

- A correct implementation satisfies both requirements: (1) the overlapped version produces the same result as the naive GPU version, and (2) the time is significantly reduced compared to the native offload version, but is similar to that of the naive GPU version.

5. Extend your script to also support overlapped offloading in the backward pass.

   Hint:

   - You can use `torch.autograd.Function` to implement customized backward passes for modules.
   - Training additionally consumes GPU memory for storing gradients and optimizer states (e.g., moving averages of gradients in Adam). You also need to change the implementation of `torch.optim.Adam` to offload gradients and optimizer states.

6. Combine your forward-backward implementations to train an LLaMA model on the GPU with offloading. Use the `torch.optim.Adam` optimizer with `torch.float16` or `torch.float32` precision.

Submission

You should submit a zip file containing your code and a PDF report.

In your code, besides completing the tasks described in the **Instruction** section, you should write comments to briefly explain every function and class you have implemented.

In your report, you should first describe how to run your code. Your report should also include the following contents:

1. Show your model configuration (e.g., hidden size, number of layers, etc.).
2. How much memory is required for training/inference of your small size LLaMA model with/without parameter offloading? Fill in Table 1.

	Inference	Training
w/o offload		
w/ blocked offload		
w/ overlapped offload		

Table 1: Memory consumption.

3. How much time does one step training/inference on your model cost, with/without parameter offloading? Fill in Table 2.

	Inference	Training
w/o offload		
w/ blocked offload		
w/ overlapped offload		

Table 2: Time cost.

4. Please specify the type (e.g., Nvidia RTX 2070) and memory capacity of the GPU you are using. What is the largest model it can train using the fixed batch size?