Project Title:
Developing a Generic Model Loader for Quantization
Supervisor:
Dr. Antoni Viros i Martin
Team Member (3):
Jian Zhang(jz3607), Xinchen Zhang(xz3052), Hairan Rodger Liang(hrl2121)

Abstract:

This project aims to develop a generic model loader capable of handling multiple quantization formats, such as FP8 and GPTQ, ensuring efficient memory usage. The loader will support large-scale models distributed across multiple GPUs, handling complex memory management challenges while maintaining performance. Although regular GPUs can be used for most testing, specialized hardware (e.g., L40s or H100) will be required to fully validate FP8-specific checkpoints. The model loader will also integrate with IBM FMS for testing and validation across various models and architectures.

Goal/Objective:

The primary objective is to create a memory-efficient and scalable model loader that can support multiple quantization schemes. The loader will:

1. Optimize memory usage to handle large-scale models efficiently across multi-GPU environments.
2. Support multiple quantization formats including FP8 and GPTQ, ensuring compatibility and byte alignment.
3. Integrate with IBM FMS for testing and performance evaluation.
4. Ensure scalability by maintaining correct memory alignment and byte-matching across various quantization schemes.

Challenges:

1. Memory Optimization: Quantized models, particularly those using FP8, require precise memory alignment. Ensuring that large models are loaded and executed efficiently across multiple GPUs without excessive overhead will be a challenge.
2. Hardware Constraints: While general-purpose GPUs will be used for most tests, specialized hardware (e.g., L40 or H100 GPUs) is necessary for full FP8 validation, which may limit the ability to fully test certain aspects of the loader.
3. Quantization Compatibility: Ensuring that the loader supports multiple formats without excessive performance degradation is challenging, especially when handling formats with different byte alignment and memory requirements.

Approach and Performance Optimization Techniques:

1. Memory Management Techniques:

- Lazy Loading: Load model weights incrementally as needed to minimize peak memory usage.
- Quantization-aware Memory Allocation: Use byte-level memory alignment to handle different quantization formats efficiently.

- Shared Memory Access: Optimize memory usage by allowing multiple GPUs to share access to common data during model inference, so neither of them are overwhelmed.

2. Multi-GPU Scaling Strategies:

- Data Parallelism: Implement data-parallel model training and inference to distribute workloads across GPUs, as learned in lecture.
- Pipeline Parallelism: Break model operations into segments to pipeline across multiple GPUs to reduce memory footprint per GPU (also learned in lecture).

3. Quantization Format Handling:

- Dynamic Byte Matching: Ensure proper handling of quantized model formats like FP8, with byte alignment at the model checkpoint loading stage.
- Format-Specific Optimizations: Adjust memory layout and computations for each quantization format to improve performance.

Implementation Details:

- Use Python and PyTorch for implementation, starting from existing loader frameworks and adding custom modules for handling quantization formats.
- Validate memory alignment using synthetic models and track how different quantization formats impact performance.
- Conduct tests on multiple GPUs using cloud-based platforms (e.g., GCP, IBM Cloud) for benchmarking.

Hardware - GPUs like L40 or H100: Testing across available hardware for memory alignment and loading efficiency.
Software - Cuda/PyTorch/TensorFlow: The loader will be implemented using deep learning frameworks like Cuda C, PyTorch and TensorFlow due to their support for quantization techniques and multi-GPU operations.
Dataset:
- FP8-Specific Checkpoints: Custom datasets and model checkpoints specific to FP8 will be utilized where necessary to validate quantization format compatibility.
- IBM FMS Datasets: FMS-provided models will be used to evaluate performance https://huggingface.co/ibm-fms

Demo Planned:

- A functional generic model loader that supports FP8, GPTQ, and other formats.
- Performance benchmarks comparing efficiency across different quantizations
- A demonstration showing the loader's performance with IBM FMS models, highlighting memory management improvements.

References:

- IBM FMS documentation: https://huggingface.co/ibm-fms

- CUDA memory management guide: https://docs.nvidia.com/cuda/
- PyTorch multi-GPU support: Optional: Data Parallelism — PyTorch Tutorials 2.5.0+cu124 documentation