Reimplementing a GPU-Specialized Parameter Server for Distributed PyTorch Training

Aditya Tejpaul (2021111025), Akshat Sanghvi (2021101094), Keshav Gupta (2021101018)

Overview

This project aims to reimplement a distributed parameter server library inspired by the GeePS framework and extend a typical single-GPU PyTorch training setup to support data-parallel distributed training across multiple GPUs. By integrating efficient GPU memory management, batched data operations, and background communication, the proposed system will enable scalable deep learning and overcome the limitations of single-device training.

Background and Motivation

Modern deep learning models demand substantial computational resources and fast data movement, especially when scaling to large datasets or complex architectures. The GeePS framework demonstrates how a GPU-specialized parameter server can overcome bottlenecks in distributed deep learning by:

- **Managing GPU Memory Efficiently:** Keeping parameter caches in GPU memory and overlapping data transfers with computation.
- **Batch-based Data Operations:** Using pre-built indices and batched read/update operations to maximize parallelism.
- **Optimizing Synchronization:** Supporting Bulk Synchronous Parallel (BSP) execution to achieve fast convergence despite distributed settings.

While PyTorch provides robust single-GPU training workflows, extending these to multi-GPU distributed systems often requires non-trivial modifications. This project will bridge that gap by reimplementing the core ideas of GeePS and

integrating them with PyTorch, thereby enabling efficient data-parallel training on distributed GPU clusters.

Objectives

1. **Reimplementation of the Parameter Server Library:**
    - Develop a new implementation of a GPU-specialized parameter server based on the design principles from GeePS.
    - Support batched read and update operations, GPU-CPU memory management, and background data transfers.
    - Implement synchronization mechanisms (e.g., BSP, with potential exploration of SSP variants) tailored for deep neural network training.
2. **Extension of PyTorch Training Pipeline:**
    - Integrate the reimplemented parameter server with a typical single-GPU PyTorch training loop.
    - Design a distributed data-parallel module that seamlessly replaces or augments PyTorch's native DataParallel interface.
    - Ensure minimal changes to user code while providing substantial performance improvements on multi-GPU setups.
3. **Performance Evaluation and Scalability:**
    - Benchmark the distributed training framework on standard datasets (e.g., CIFAR-10, ImageNet subsets) and models.
    - Measure improvements in training throughput, convergence time, and GPU memory utilization relative to single-GPU training.
    - Analyze the trade-offs between synchronization strictness (BSP vs. SSP) and overall performance.

System Architecture and Design

Parameter Server Module

- **GPU Memory Management:** Implement a mechanism to maintain a parameter cache in GPU memory, with support for dynamic swapping between GPU and CPU memory.
- **Batched Operations:** Design APIs for batched read and update operations that leverage pre-built indices to optimize data access on GPU.
- **Background Data Movement:** Utilize dedicated threads or CUDA streams to perform asynchronous data transfers between CPU and GPU, overlapping communication with computation.

PyTorch Integration

- **Distributed Data-Parallel Module:** Develop a wrapper around the PyTorch training loop that interfaces with the parameter server for parameter synchronization and gradient updates.
- **Minimal Intrusiveness:** Ensure that existing single-GPU training scripts can be adapted with minimal modifications, preserving the ease-of-use and flexibility of PyTorch.
- **Synchronization Strategy:** Start with a BSP model to ensure convergence quality, with an option to explore relaxed consistency models in later iterations.

Implementation Plan

1. **Development of the Parameter Server Library:**
    - Code the core parameter server in C++/CUDA for efficient low-level GPU operations.
    - Implement and test batched parameter read/update functions and background memory management.
2. **Integration with PyTorch:**
    - Develop Python bindings for the parameter server using PyBind11 or a similar framework.
    - Create a custom PyTorch module that leverages these bindings to handle distributed training.
3. **Testing and Evaluation:**

- Set up experiments to benchmark the performance and scalability of the distributed training framework.
- Compare the extended distributed training pipeline against a baseline single-GPU PyTorch implementation on various models and datasets.

4. **Documentation and Final Reporting:**
    - Document the implementation details, usage instructions, and experimental results.
    - Prepare a final report and presentation summarizing the contributions and performance benefits.

Timeline

- **Day 1–10:** Implementation of the GPU-specialized parameter server.
- **Day 10–19:** Integration with PyTorch and development of distributed training module.
- **Day 19–27:** Testing, performance evaluation, and iterative optimization.
- **Day 27-33:** Documentation, final report preparation, and project presentation.

Conclusion

This project offers an exciting opportunity to explore cutting-edge distributed systems techniques within the context of deep learning. By reimplementing a state-of-the-art parameter server and integrating it with PyTorch, the project will not only advance academic understanding of data-parallel training on GPUs but also provide practical tools that can enhance the scalability and efficiency of deep learning applications.