HPC-DL: High-Performance Deep Learning Kernels and Distributed Systems Architecture

Anshuman Agrawal
`iemAnshuman/high-performance-deep-learning`

February 9, 2026

Abstract

This report details the implementation and performance analysis of `HPC-DL`, a high-performance library designed to bypass standard PyTorch bottlenecks in deep learning workloads. The project focuses on optimizing memory bandwidth utilization through custom OpenAI Triton kernels, implementing distributed communication primitives via MPI, and enabling system-level INT4 quantization for efficient inference. Key results include a 3.53x speedup in Softmax operations and the successful deployment of a Ring All-Reduce algorithm for distributed gradient synchronization.

1 Introduction

As deep learning models scale to billions of parameters, the primary bottleneck in hardware acceleration has shifted from compute capability to memory bandwidth. Standard eager execution frameworks often suffer from excessive High Bandwidth Memory (HBM) round-trips. This project investigates these bottlenecks by benchmarking hardware limits and implementing custom kernels and distributed systems primitives to optimize data movement.

2 Kernel Fusion and Optimization

To mitigate HBM latency, custom fused kernels were implemented using OpenAI Triton, keeping intermediate activations in SRAM (Static Random Access Memory).

2.1 Fused Softmax

A custom Softmax kernel was developed to address numerical instability and memory inefficiency. The implementation utilizes "Safe Softmax" with online normalization, where the maximum value of a row is subtracted to prevent floating-point overflow ($y_i = \frac{e^{x_i - m}}{\sum e^{x_j - m}}$).

- **Memory Strategy:** The kernel loads entire rows into SRAM to maximize memory coalescing and performs reductions (Max and Sum) in registers, avoiding round-trips to global memory.
- **Performance:** Profiling reveals a 3.53x speedup over the PyTorch baseline, with memory throughput reaching 84.32% utilization.

2.2 Fused MLP

The Multi-Layer Perceptron (MLP) implementation fuses the Bias-Add and GeLU activation functions. It employs an inline GeLU approximation using a manual Tanh implementation to ensure robustness across different Triton versions.

3 Distributed Training Primitives

Scalable training requires efficient inter-node communication. This project implements fundamental primitives in C++ using MPI.

3.1 Ring All-Reduce

A Ring All-Reduce algorithm was implemented to optimize bandwidth usage during gradient synchronization. The logic partitions vectors into N chunks (where N is the world size) and circulates them through the ring topology.

- **Deadlock Prevention:** An ”Even/Odd” strategy was adopted where even ranks send then receive, and odd ranks receive then send, preventing blocking conditions common in naive implementations.

3.2 Gradient Bucketing

To hide network latency during the backward pass, a custom Distributed Data Parallel (DDP) wrapper was implemented. It aggregates gradients into optimal 25MB buckets before synchronization, reducing the overhead of small tensor transmissions.

4 System-Level Quantization and Inference

4.1 INT4 Quantization

A custom C++ extension was developed to perform bit-packing, compressing two 4-bit integers into a single `uint8` container. This approach reduces the memory footprint of model weights by approximately 8x compared to FP32.

4.2 Zero-Copy Inference

To enable the loading of models larger than available physical RAM, a `mmap`-based loader was implemented. This allows tensors to be mapped directly from disk to virtual memory, leveraging the OS page cache to fault in data only when accessed.

5 Performance Analysis

A Roofline Analysis was conducted to categorize kernels as either Compute Bound or Memory Bound.

Kernel	**Arithmetic Intensity**	**Bottleneck**	**Performance**
Vector Add	0.08 FLOPs/Byte	Memory (HBM)	19.91 GFLOPS
MatMul (4096^2)	682.67 FLOPs/Byte	Compute (Tensor Core)	4374.86 GFLOPS
Fused Softmax	High	L2 Cache / SRAM	3.53x Speedup

Table 1: Roofline Analysis Summary.

The analysis confirms that element-wise operations like Vector Add are severely memory-bound, justifying the need for kernel fusion to increase arithmetic intensity.

6 Conclusion

The `HPC-DL` library demonstrates that significant performance gains can be achieved by bypassing high-level framework abstractions. By fusing kernels to utilize SRAM and implementing low-level distributed primitives, the project successfully mitigates memory bandwidth bottlenecks inherent in large-scale deep learning workloads.