High-Performance GEMM: From Kernel Optimization to Tensor Parallelism

Aryan Jain[*] **Fanyi Pu**[*] **Ze Hong Maxwell Au**[*]
School of Computer Science and Engineering
Nanyang Technological University, Singapore
`{ARYAN017, FPU001, MAU002}@e.ntu.edu.sg`

Abstract

General Matrix–Matrix Multiplication (GEMM) serves as the computational cornerstone of modern deep learning. This project bridges the gap between *kernel-level optimization* and *system-level parallelism* by (i) implementing and progressively optimizing CUDA GEMM kernels on a single GPU, and (ii) extending these primitives to a multi-GPU Tensor Parallel linear layer utilizing NCCL-based communication. We systematically evaluate the compute–communication trade-offs, analyze strong/weak scaling behaviors, and quantify the impact of kernel efficiency on distributed training performance.

1 Introduction

General Matrix Multiplications (GEMM) underpins virtually all compute-intensive operations in deep learning [Jia et al., 2018]. In transformer architectures [Vaswani et al., 2017], multi-head attention mechanisms and feed-forward networks rely heavily on large-scale matrix multiplications, establishing GEMM performance as a primary determinant of system throughput. As model parameters scale beyond the memory capacity of a single accelerator, **Tensor Parallelism** [Shoeybi et al., 2019] has emerged as a standard technique for distributing weights across GPUs, introducing inter-GPU communication as a critical bottleneck.

While vendor-optimized libraries such as cuBLAS [NVIDIA, 2024] provide highly tuned primitives, a deep understanding of the interplay between kernel-level compute efficiency and system-level communication overhead remains essential for system designers. This project aims to: (1) systematically optimize GEMM on a single GPU using CUDA; (2) implement a distributed Tensor Parallel linear layer (forward and backward) using NCCL; and (3) quantitatively analyze how local kernel optimization interacts with distributed scaling efficiency.

2 Proposed methodology

Single-GPU GEMM optimization. We develop a high-performance GEMM kernel from scratch in CUDA, following a progressive optimization roadmap: (1) **naive implementation** utilizing global memory with coalesced access; (2) **tiled GEMM with shared memory** to maximize data reuse and locality; (3) **register blocking and loop unrolling** to enhance arithmetic intensity; and (4) **Tensor Core acceleration** (optional) via WMMA intrinsics for mixed-precision performance. At each stage, we utilize NVIDIA Nsight Compute to profile occupancy, memory bandwidth, and compute throughput (GFLOP/s), benchmarking against cuBLAS. We employ *roofline model* analysis [Williams et al., 2009] to characterize performance bottlenecks and the transition from memory-bound to compute-bound regimes.

[*]Equal contribution. Authors listed in alphabetical order.

Multi-GPU Tensor Parallel forward pass. We implement column-wise Tensor Parallelism for a linear layer. Given $Y = XW$, the weight matrix is partitioned column-wise across p GPUs: $W = [W_1, \dots, W_p]$, such that GPU i computes $Y_i = XW_i$. Partial results are gathered via an NCCL `AllGather` operation. We investigate techniques for overlapping communication with computation using CUDA streams, analyze the impact of partition granularity, and quantify communication overhead as a function of GPU count.

Backward pass and gradient aggregation. During the backward pass, each GPU computes local gradients: $\partial\mathcal{L}/\partial W_i = X^\top(\partial\mathcal{L}/\partial Y_i)$ and $\partial\mathcal{L}/\partial X = \sum_{i=1}^{p}(\partial\mathcal{L}/\partial Y_i)W_i^\top$. The computation of $\partial\mathcal{L}/\partial X$ requires an `AllReduce` operation to aggregate gradients across GPUs. We compare synchronous execution against computation-overlapped reduction to demonstrate how communication latency becomes the dominant factor as local GEMM kernels are optimized.

3 Evaluation plan

We conduct all experiments on NVIDIA A100 GPUs.

Single-GPU performance. We measure throughput (GFLOP/s) across a range of matrix dimensions (512–8192) for each optimization stage, benchmarking against cuBLAS. Roofline analysis will be used to visualize the optimization trajectory.

Strong and weak scaling. For **strong scaling**, we fix the total problem size and increase the number of GPUs ($1, 2, 4, 8$) to measure speedup and parallel efficiency. For **weak scaling**, we fix the workload per GPU to evaluate linear scalability.

Compute–communication trade-off. We quantify the ratio of wall-clock time spent in NCCL communication versus GEMM computation. This metric allows us to evaluate how aggressive kernel optimization shifts the system bottleneck from computation to communication.

4 Expected contributions

Key contributions of this project include: (1) a comprehensive study of CUDA GEMM optimization driven by profiling and roofline analysis; (2) a functional implementation of a multi-GPU Tensor Parallel linear layer (forward and backward) using NCCL; (3) a quantitative analysis of compute–communication trade-offs and scaling characteristics; and (4) insights into the dependency of distributed scaling efficiency on underlying kernel performance.

References

Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele P Scarpazza. Dissecting the NVIDIA Volta GPU architecture via microbenchmarking. *arXiv preprint arXiv:1804.06826*, 2018.

NVIDIA. cuBLAS library. `https://developer.nvidia.com/cublas`, 2024.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.