SC4064 Proposal

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) is the computational backbone of modern deep learning. This project bridges *kernel-level optimization* and *system-level parallelism* by (i) implementing and progressively optimizing CUDA GEMM kernels on a single GPU, and (ii) extending to a multi-GPU Tensor Parallel linear layer with NCCL-based communication. We systematically evaluate the compute–communication trade-off, strong/weak scaling behavior, and the impact of kernel-level optimization on distributed scaling efficiency.

1 Introduction

General Matrix–Matrix Multiplication (GEMM) underpins virtually all compute-intensive operations in deep learning [Jia et al., 2018]. In transformer architectures [Vaswani et al., 2017], multi-head attention and feedforward layers reduce to large matrix multiplications, making GEMM performance a first-order concern. As model sizes grow beyond the memory capacity of a single accelerator, **Tensor Parallelism** [Shoeybi et al., 2019] has become standard for distributing parameters across GPUs, where inter-GPU communication becomes a critical throughput bottleneck.

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

2 Proposed methodology

Single-GPU GEMM optimization. We implement GEMM from scratch in CUDA and progressively optimize: (1) **naive implementation** using global memory with coalesced access; (2) **tiled GEMM with shared memory** to exploit data locality; (3) **register blocking and loop unrolling** to increase arithmetic intensity; (4) **Tensor Core acceleration** (optional) via WMMA intrinsics for mixed-precision computation. At each stage, we profile using NVIDIA Nsight Compute to measure occupancy, bandwidth utilization, and throughput (GFLOP/s), benchmarking against cuBLAS. We employ *roofline model* analysis [Williams et al., 2009] to classify each variant as compute-bound or memory-bound.

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: $W = [W_1, \ldots, W_p]$, where GPU i computes $Y_i = XW_i$. Outputs are gathered via NCCL `AllGather`. We investigate communication–computation overlap using CUDA streams, the effect of partition granularity, and communication overhead as a function of GPU count.

*Equal contribution. Authors listed in alphabetical order.

Backward pass and gradient aggregation. Each GPU computes local gradients: $\partial\mathcal{L}/\partial W_i = X^\top(\partial\mathcal{L}/\partial Y_i)$ and $\partial\mathcal{L}/\partial X = \sum_{i=1}^{p}(\partial\mathcal{L}/\partial Y_i)W_i^\top$, where the latter requires an `AllReduce` across GPUs. We compare synchronous versus computation-overlapped reduction, revealing how communication cost dominates as local GEMM becomes highly optimized.

3 Evaluation plan

All experiments will be conducted on NVIDIA A100 GPUs.

Single-GPU performance. We measure throughput (GFLOP/s) across matrix sizes (512–8192) for each optimization stage, comparing against cuBLAS. Roofline analysis identifies bottleneck transitions between memory-bound and compute-bound regimes.

Strong and weak scaling. For strong scaling, we fix the problem size and increase GPUs $(1, 2, 4, 8)$, measuring speedup and parallel efficiency. For weak scaling, we fix per-GPU workload and evaluate linear scalability.

Compute–communication trade-off. We measure the proportion of wall-clock time in NCCL communication versus GEMM computation, evaluating how kernel optimization shifts the bottleneck from compute to communication.

4 Expected contributions

This project delivers: (1) a profiling-driven study of CUDA GEMM optimization with roofline analysis; (2) a working multi-GPU Tensor Parallel linear layer (forward and backward) using NCCL; (3) quantitative analysis of compute–communication trade-offs and scaling behavior; and (4) insights into how kernel-level optimization interacts with distributed parallelism.

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.