SC4064 GPU Programming Project Proposal: Scaling GEMM from Single-Kernel Optimization to Multi-GPU 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 Multiplication (GEMM) is the computational backbone of modern deep learning. This project explores the synergy between *low-level CUDA kernel optimization* and *system-level distributed parallelism*. We aim to (i) progressively optimize custom CUDA GEMM kernels on a single GPU to approach theoretical peak performance, and (ii) extend these optimized primitives to a multi-GPU Tensor Parallel linear layer using NCCL-based communication. By analyzing the interplay between hardware-aware kernel efficiency and inter-node communication, we provide a quantitative study of strong/weak scaling behaviors and identify the shifting bottlenecks in distributed GPU systems.

1 Introduction

General Matrix Multiplication (GEMM) operations underpin virtually all compute-intensive workloads in deep learning [Jia et al., 2018]. In Transformer-based architectures [Vaswani et al., 2017], the performance of multi-head attention and feed-forward networks is primarily dictated by the efficiency of underlying CUDA kernels. As model scales exceed the memory capacity of individual accelerators, **Tensor Parallelism** [Shoeybi et al., 2019] has become an indispensable technique for distributed training, albeit at the cost of introducing significant inter-GPU communication overhead.

While vendor-tuned libraries like cuBLAS [**?**] offer near-optimal performance, they often abstract away the complex interaction between hardware-level compute intensity and system-level communication latency. For GPU programming practitioners, understanding these low-level details is crucial. This project seeks to bridge this gap by: (1) systematically optimizing custom GEMM kernels to understand hardware-level constraints such as memory coalescing and bank conflicts; (2) implementing a distributed Tensor Parallel linear layer (forward and backward) using NCCL; and (3) quantifying how the efficiency of local CUDA kernels impacts the overall scalability and speedup of multi-GPU systems.

2 Proposed Methodology

Single-GPU GEMM Optimization. We will implement and refine a CUDA GEMM kernel through a hierarchical optimization roadmap, focusing on hardware-aware programming: (1) **Naive Global Memory Access**: Establishing a baseline with coalesced memory patterns; (2) **Shared Memory Tiling**: Minimizing global memory traffic by leveraging on-chip memory and addressing bank conflicts; (3) **Register Blocking & Instruction Parallelism**: Increasing arithmetic intensity through register-level data reuse and loop unrolling; (4) **Tensor Core Integration** (Optional): Utilizing WMMA (Warp-Level Matrix Operations) intrinsics for mixed-precision acceleration. Performance

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

will be characterized using the *Roofline Model* [Williams et al., 2009] and profiled via NVIDIA Nsight Compute to analyze occupancy and throughput (TFLOPS) relative to cuBLAS.

Multi-GPU Tensor Parallelism. We will implement and compare two fundamental forms of Tensor Parallelism (TP) for the linear layer, reflecting state-of-the-art distributed training strategies [Shoeybi et al., 2019]:

- **Column Parallelism**: The weight matrix W is partitioned column-wise: $W = [W_1, \ldots, W_p]$. Each GPU i computes a partial output $Y_i = XW_i$. This is typically used for the first linear layer in a block (e.g., $h \to 4h$).
- **Row Parallelism**: The weight matrix W is partitioned row-wise: $W = [W_1; \ldots; W_p]^\top$. Each GPU i computes $Y_i = X_i W_i$, where X_i is a column slice of the input. This is used for the subsequent layer (e.g., $4h \to h$).

By strategically combining these two patterns, we can implement a complete Parallel MLP block that requires only a single NCCL `AllReduce` operation at the end of the block, significantly reducing synchronization overhead. We will focus on:

- **Forward Pass**: Implementing NCCL `AllReduce` and `AllGather` primitives, with an emphasis on overlapping communication with GEMM computation using asynchronous CUDA streams.
- **Backward Pass**: Managing gradient synchronization and analyzing the overhead of aggregating $\partial\mathcal{L}/\partial X$ across multiple GPUs.

We will analyze how the communication-to-computation ratio evolves as our custom CUDA GEMM kernels are optimized.

3 Evaluation Plan

Experiments will be conducted on a cluster of NVIDIA A100 GPUs (via NSCC or local lab resources).

Kernel Benchmarking. We will measure GFLOPS across various matrix sizes (512 to 16384) to identify the transition from memory-bound to compute-bound regimes for each CUDA optimization stage.

Scaling Analysis. We will evaluate **Strong Scaling** (fixed total workload, increasing GPUs) to measure parallel efficiency, and **Weak Scaling** (fixed workload per GPU) to assess the system's ability to handle larger models.

Bottleneck Identification. By comparing the execution time of NCCL collectives against our custom CUDA kernels, we will quantify the "crossover point" where further kernel optimization yields diminishing returns due to communication dominance.

4 Expected Contributions

The project expects to deliver: (1) A suite of custom CUDA GEMM kernels with documented performance gains at each optimization stage; (2) A functional NCCL-based Tensor Parallel linear layer implementation; (3) A comprehensive technical report analyzing the trade-offs between hardware-level CUDA optimization and system-level scaling efficiency.

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.

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, pages 5998–6008, 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.