Overlapping Gradient Computation and Communication in HPC Training Loops

Authors: Sumukha Madodi, Shraddha Singh, Shashank M Bharadwaj, Sahana Diwakar
Course: ECE 759 – High Performance Computing
Semester: Spring 2025
University of Wisconsin–Madison
https://github.com/Sumukha-M/proj_759

Abstract

This project explores methods to overlap gradient computation and communication in distributed deep learning workflows, using CUDA, OpenMP, and MPI. We implemented a multi-phase system simulating various forms of compute-comm parallelism, starting with basic CUDA-based training, followed by CPU-based task parallelism, hybrid MPI-GPU simulation, and a final performance-focused simulation. Through chunked processing and multithreaded designs, we quantified the time savings from overlapping using Gantt charts, execution profiling, and performance plots.

1. Introduction

Deep learning training involves large amounts of computation followed by communication—particularly gradient aggregation across devices in data-parallel setups. In HPC systems, communication can easily become a bottleneck if done sequentially. This project investigates overlapping strategies to reduce end-to-end training time by concurrently executing gradient computation and inter-node communication using CUDA, OpenMP, and MPI constructs.

2. Motivation and Objectives

As neural network models scale, training becomes increasingly compute- and bandwidth-intensive. Overlapping communication with computation can minimize idle time and improve resource utilization. This project simulates realistic overlapping behavior, quantifies its benefits, and provides insights into how chunking and scheduling can influence performance.

3. Phase-wise Implementation

3.1 matrix_train_cuda: CUDA Baseline Training

In this phase, we implemented a training loop with CUDA. It involved forward and backward matrix multiplication and gradient computation on a fixed-size input and weight matrix (1024×512, 512×512). A CUDA kernel computed the output `y = x * w` and then a backward pass calculated the gradient of the loss w.r.t. the weights using `grad_w = x^T * grad_out`. The weights were updated with a standard SGD step.

All computation was done on GPU using unified memory. Kernel launches were separated and synchronized after each step to reflect non-overlapped baseline training. This served as the performance baseline for further phases.

3.2 cpu_parallel_overlap: Simulated Overlap on CPU with OpenMP

In this phase, we simulated overlapping using OpenMP parallel sections. One thread handled simulated compute (e.g., `compute_gradients()`), and another handled communication (e.g., `communicate_gradients()`). Execution time per chunk was simulated using timed sleep (`std::this_thread::sleep_for`) to mimic real workloads.

The sequential and parallel versions were compared by toggling OpenMP off/on. We recorded execution time differences and generated plots: a Gantt-style timeline and bar chart comparing total execution time. This validated the theoretical benefit of compute-comm overlap even in simulated CPU environments.

3.3 cuda_mpi_overlap: Mock MPI + CUDA Overlap

In this stage, we introduced MPI (simulated with mpiexec) to represent a multi-rank environment and paired it with OpenMP. Each process used OpenMP sections to simulate a forward/backward pass and a mock `MPI_Allreduce` (delayed with sleep). Each rank operated on its own gradient vector. The overlap of compute and MPI communication was handled using OpenMP sections.

This helped us understand synchronization challenges and timing behavior in distributed systems even without real GPU clusters. It prepared the ground for asynchronous or NCCL-like overlapping in Phase 4.

3.4 overlap_sim: Chunked Overlapping Simulation and Profiling

The final phase implemented a chunked simulation with overlapping compute and communication per chunk using C++ threads. Each chunk simulated 300–400 ms of computation and 150–200 ms of communication.

We stored per-chunk timings and plotted:
- Chunk execution consistency
- Total execution time comparison
- Scaling behavior across number of chunks (5 to 50)

- Gantt chart timeline view of overlapping

The program also output a CSV log of timing, which was parsed by Python scripts to create detailed performance plots.

4. Optimization Strategies

- Used OpenMP sections for lightweight thread-level parallelism.
- Reduced thread contention by ensuring compute and comm run in isolation.
- Simulated chunk processing to better match gradient layers in real models.
- Automated compilation using Makefiles (with targets for both sequential and overlap mode).
- Separated modular files (compute.cpp, communicate.cpp, main.cpp) for clarity and reusability.

5. Challenges and Resolutions

- Euler HPC environment posed MPI linking issues → we pivoted to local simulation.
- CUDA stream-level overlap wasn't implemented due to system constraints, but simulated logic captured the intent.
- Ensured accurate timing despite OS scheduling noise by simulating multiple chunks.
- Balanced complexity with visual clarity to keep plots intuitive.

6. Results and Analysis

Figure 1: Timeline visualization of overlapping vs sequential execution.

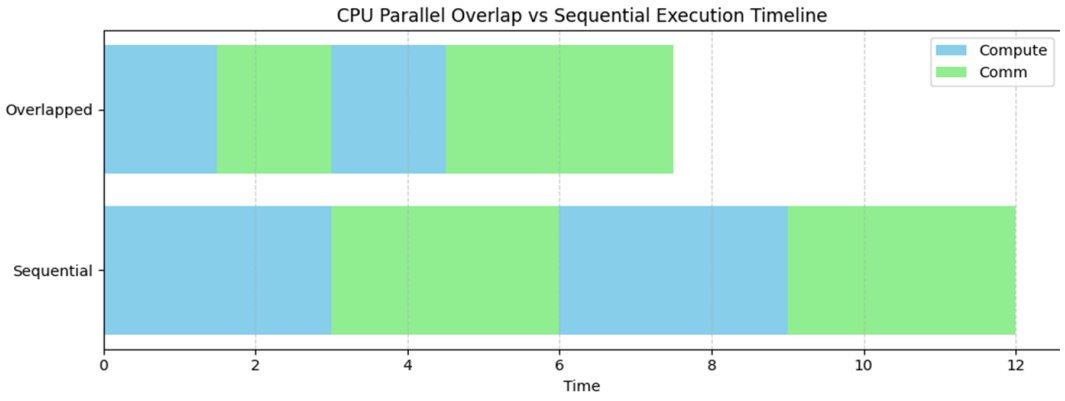


Figure 2: Chunk-wise execution time showing consistent overlap performance.

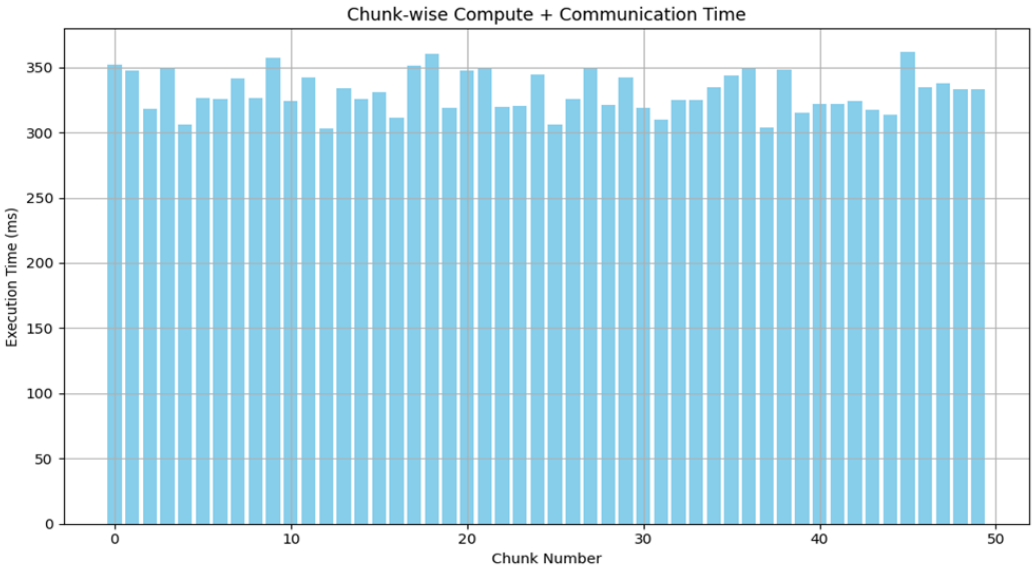


Figure 3: Bar chart comparing total time with and without overlap.

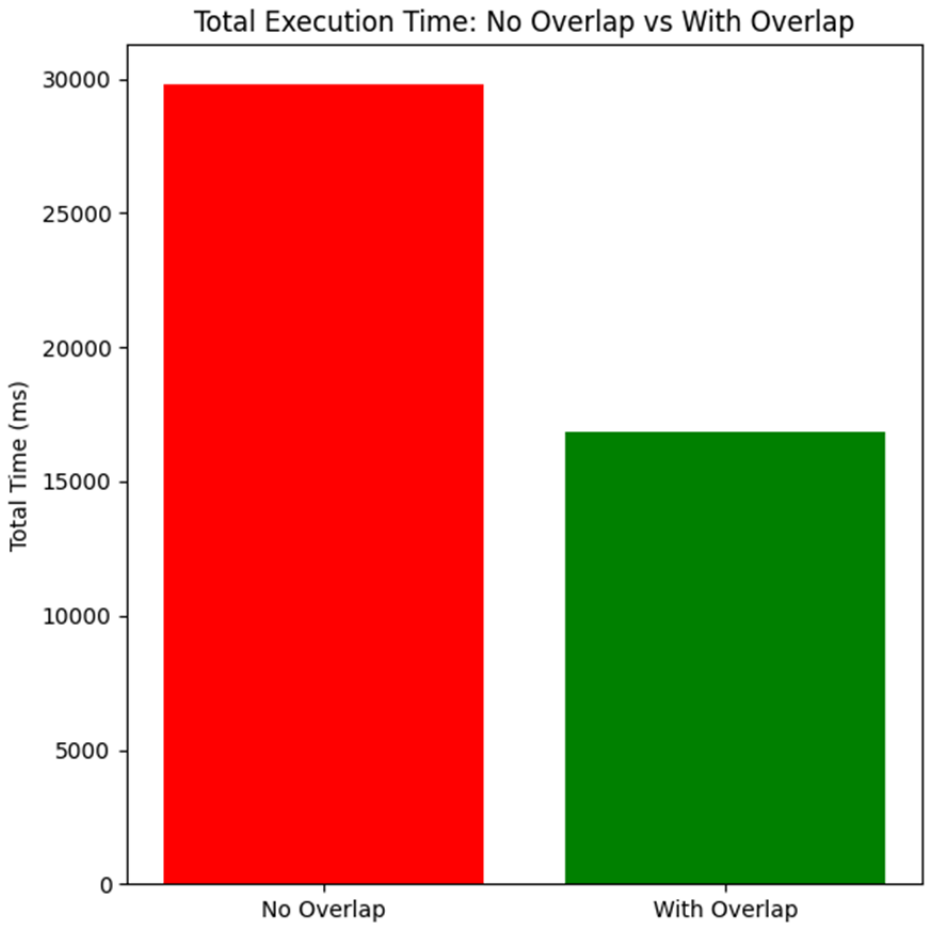
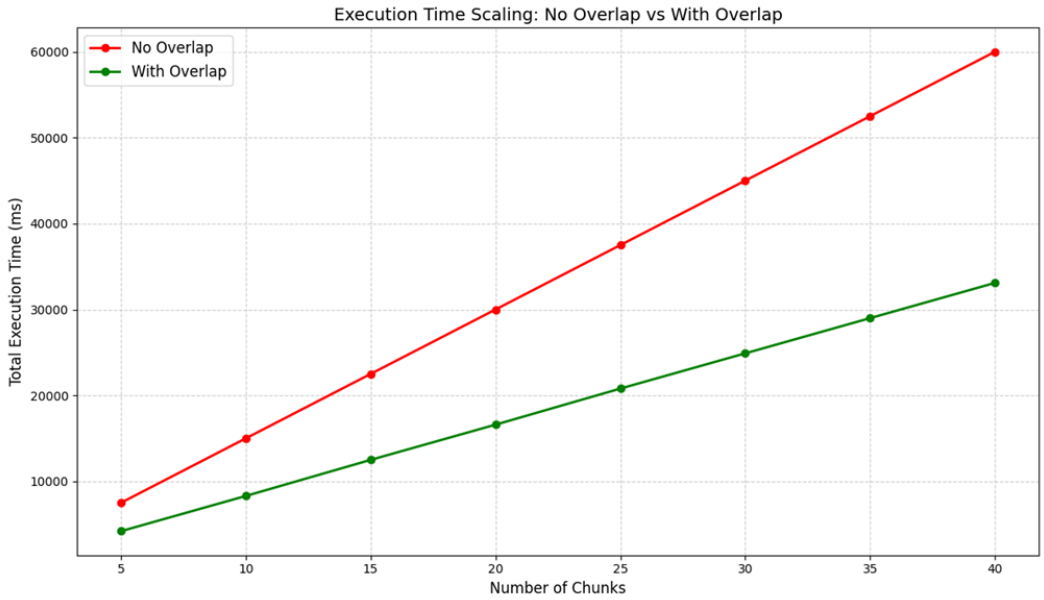


Figure 4: Line chart showing execution time scaling with increasing chunk count.



7. Conclusion

This project demonstrated the effectiveness of overlapping gradient computation and communication using simulated and real GPU workflows. We showed measurable performance improvements through chunked simulation and OpenMP-based parallelism. The structured progression from CUDA baseline to MPI simulation allowed us to understand and evaluate HPC design trade-offs effectively.

Future work could include real multi-GPU validation with NCCL or Horovod, hybrid pipeline/data parallelism, and integration into real training frameworks.