High-Performance Computing Project Report: Neural Network Acceleration on GPUs for MNIST Classification

Ali Haider (22i-1210)
Awais Khan (22i-0997)
Muhammad Shayan Memon (22i-0773)

Course: High-Performance Computing
Date: April 20, 2025

GitHub Repository:
https://github.com/A-l-i-h-a-i-d-e-r/HPC*Project.git*

---

1. Introduction

The MNIST classification problem is a benchmark in machine learning, involving the recognition of handwritten digits (0–9) from a dataset of 70,000 grayscale images (28x28 pixels). This project focuses on accelerating a neural network implementation for MNIST classification using CUDA on GPUs. The goal is to optimize a baseline sequential CPU implementation (V1) through progressive GPU-based versions (V2, V3, V4), leveraging parallel computing techniques to enhance performance while maintaining accuracy.

The project evaluates four implementations:

- **V1**: Sequential CPU implementation.
- **V2**: Naive GPU implementation using CUDA.
- **V3**: Optimized GPU implementation with dynamic launch configurations, memory optimizations, and asynchronous streams.

- **V4**: Advanced GPU implementation incorporating Tensor Cores via cuBLAS with TF32 precision.

This report details the methodology, optimizations, results, performance analysis, and individual contributions, with reference to the private GitHub repository for code and commit history.

2. Methodology

2.1. Neural Network Architecture

The neural network is a fully connected feedforward model with:

- **Input Layer**: 784 nodes (28x28 pixel images flattened).
- **Hidden Layer**: 128 nodes with ReLU activation.
- **Output Layer**: 10 nodes (one per digit) with softmax activation.
- **Training Parameters**: Learning rate = 0.01, epochs = 3, batch size = 64 (for V3 and V4).

2.2. Implementation Details

V1: Sequential CPU Implementation

- **Description**: Baseline implementation in C, running on CPU.
- **Key Functions**: `forward`, `backward`, `train`, `evaluate`.
- **Performance**: Executes in ˜22.377s with 96.78% test accuracy.

V2: Naive GPU Implementation

- **Description**: CUDA port of V1, parallelizing matrix and activation operations.
- **Features**:
  - CUDA kernels: `matrixMulKernel`, `reluKernel`, `softmaxKernel`, and gradients.
  - Static launch config (16x16 blocks for 2D, 256 threads for 1D).
  - Frequent host-device transfers per image.
- **Performance**: Executes in ˜183.159s, test accuracy 96.49%.

V3: Optimized GPU Implementation

- **Description**: Enhances V2 with performance-focused optimizations.
- **Optimizations**:
  - Dynamic block/grid sizes.

  - Float precision.
  - Parallel softmax using shared memory.
  - Pinned host memory.
  - Memory reuse across iterations.
  - CUDA streams for batch processing.
- **Performance**: Executes in ˜6.779s, test accuracy 96.20%.

V4: Tensor Core Implementation

- **Description**: Adds cuBLAS Tensor Core support to V3.
- **Additions**:
  - `cublasGemmEx` with TF32 precision.
  - `addBiasKernel` for bias addition.
  - Retains all V3 optimizations.
- **Performance**: Executes in ˜5.817s, test accuracy 91.93%.

2.3. Development Environment

- **Hardware**: NVIDIA GPU (Ampere+ for Tensor Core support).
- **Software**: CUDA 11.x, cuBLAS, GCC, NVCC (`-arch=sm_80`).
- **Dataset**: MNIST in `data/` as IDX files.
- **Compilation**: Makefile for V1, NVCC for V2–V4.

3. Results and Performance Analysis

3.1. Performance Metrics

Version	Time (s)	Speedup	Train Acc.	Test Acc.	Loss
V1 (CPU)	22.377	1.00x	97.86%	96.78%	0.0727
V2 (GPU)	183.159	0.12x	97.94%	96.49%	0.0699
V3 (GPU)	6.779	3.82x	97.93%	96.20%	0.0707
V4 (GPU)	5.817	4.51x	90.08%	91.93%	2.0312

Table 1: Comparison of execution time, speedup, accuracy, and loss

3.2. Analysis

- **V1**: Solid baseline with good accuracy, but slow due to sequential computation.

- **V2**: Worse performance due to poor memory handling and static kernel launches.
- **V3**: Balanced implementation with good speedup and minimal accuracy drop.
- **V4**: Fastest execution using Tensor Cores, but with noticeable accuracy trade-off.

3.3. Speedup and Trade-offs

- V3 and V4 show real benefits of GPU acceleration.
- V2's poor performance highlights the importance of memory and kernel optimization.
- TF32 in V4 reduces precision, suggesting it's more suited for deeper models.

4. Individual Contributions

- **Awais Khan**:
  - Implemented V1 and V2.
  - Developed CUDA kernels and initial GPU port.
  - Assisted in testing and debugging all versions.
- **Ali Haider**:
  - Implemented V3 with launch configs, parallel softmax, and streams.
  - Optimized memory usage.
  - Contributed to profiling and report writing.
- **Shayan Memon**:
  - Developed V4 using cuBLAS and Tensor Cores.
  - Modified forward/backward passes for cuBLAS.
  - Conducted result analysis and comparisons.

All members collaborated on the final report, presentation, and demo. Contribution logs are visible in the GitHub repository.

5. Conclusion

This project successfully accelerated MNIST classification using GPU-based CUDA techniques. Each version built upon the last, showing how proper parallelization can improve execution speed.

- **V3** is the most balanced, offering good accuracy and a 3.82x speedup.
- **V4** is the fastest but less accurate due to TF32 precision trade-offs.
- **V2** highlights that naïve CUDA implementations can be slower than optimized CPU code if not tuned properly.