High Performance Computing Project: Neural Network Acceleration on GPUs

Presenters:

Daniyal Ahmed (22I-1032)

Hammad Ali (22I-0914)

Junaid Zeb (22I-8794)

Project Overview and Objectives

Project Overview

The project focuses on accelerating MNIST digit classification using CUDA and GPU parallelism. We began with a CPU-based neural network and applied optimizations such as memory tuning, kernel configuration, minibatching, and cuBLAS tensor core acceleration.

Objectives

- Implement a baseline CPU model.
- Develop a naive CUDA version.
- Optimize with memory hierarchy and minibatching.
- Use cuBLAS for tensor cores.
- Compare performance across versions.

1 Goal

Accelerate neural network training and inference using GPUs

2 Approach

Systematically optimize each stage of the process

3 Focus

Leverage parallel processing capabilities of GPUs

1 Neural Network Overview

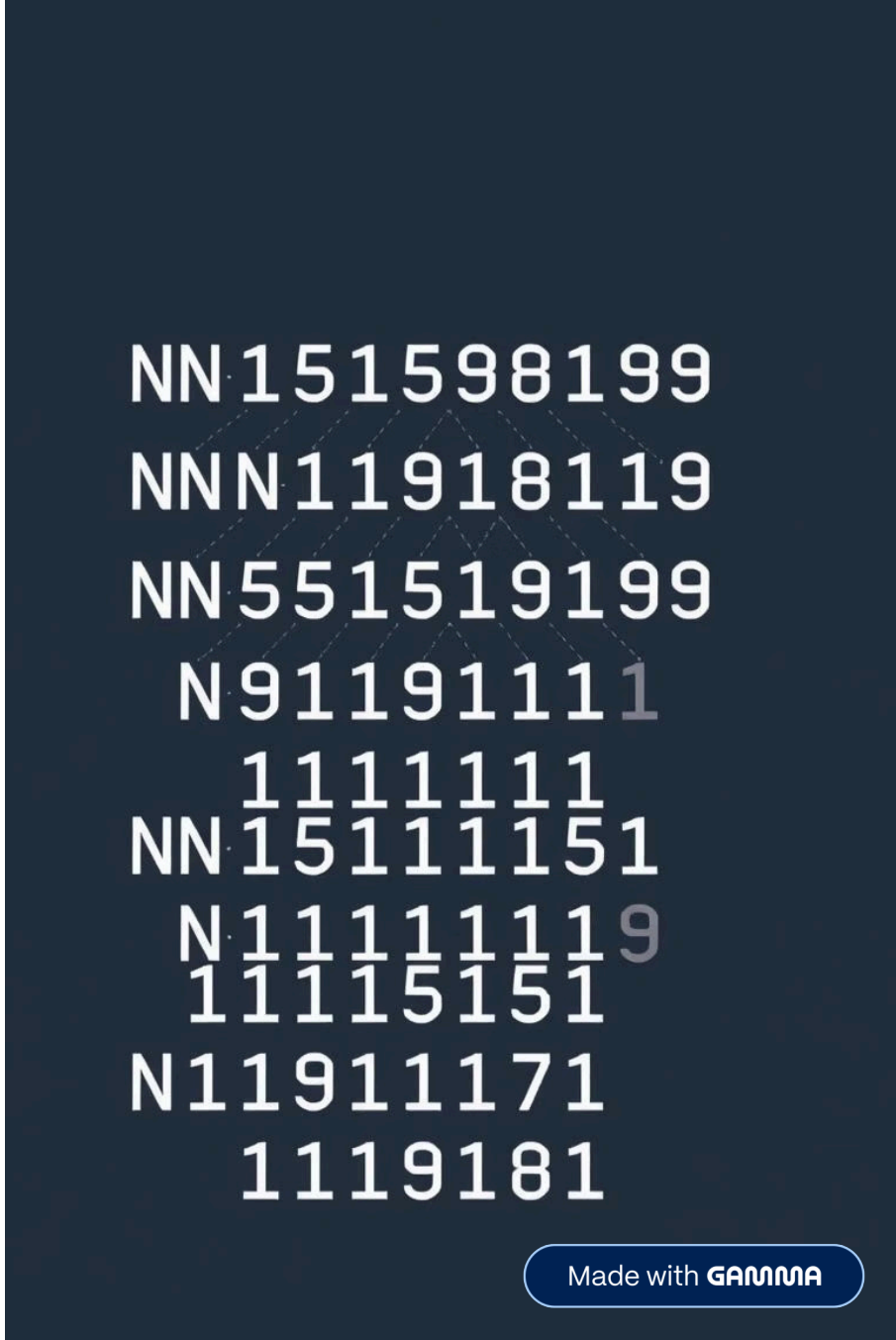
- Input Layer (784 Neurons)
- Hidden Layer (128, Relu Neurons)
- Output Layer (10, Softmax Neurons)

2 MNIST Classification

- 60,000 training images
- 10,000 test images

3 GPU Computing Basics

- CUDA enables GPU acceleration
- Optimized kernel launches
- Fast shared/local memory
- Reduced data transfer
- High thread occupancy



Implementation Versions and Optimization Techniques

1

V1: Sequential CPU

Baseline CPU implementation without any parallelism.

2

V2: Naive GPU

Initial GPU implementation with CUDA kernels for forward/backward pass.

3

V3.1: Shared Memory

Added shared memory, used local memory, and switched to float precision.

4

V3.2: Unified Kernel

Unified kernel for forward and backward passes with mini-batch gradient descent.

5

V4: Tensor Cores

Integrated tensor cores using cuBLAS for optimized GEMM operations.

Optimization Techniques

1

Memory Hierarchy Utilization

Shared memory for activations and hidden layers, local memory for per-thread scalar operations reduced global memory accesses.

2

Kernel Optimization

Thread indexing improvements, warp alignment, and conditional simplification, kernel launch configuration tuning.

3

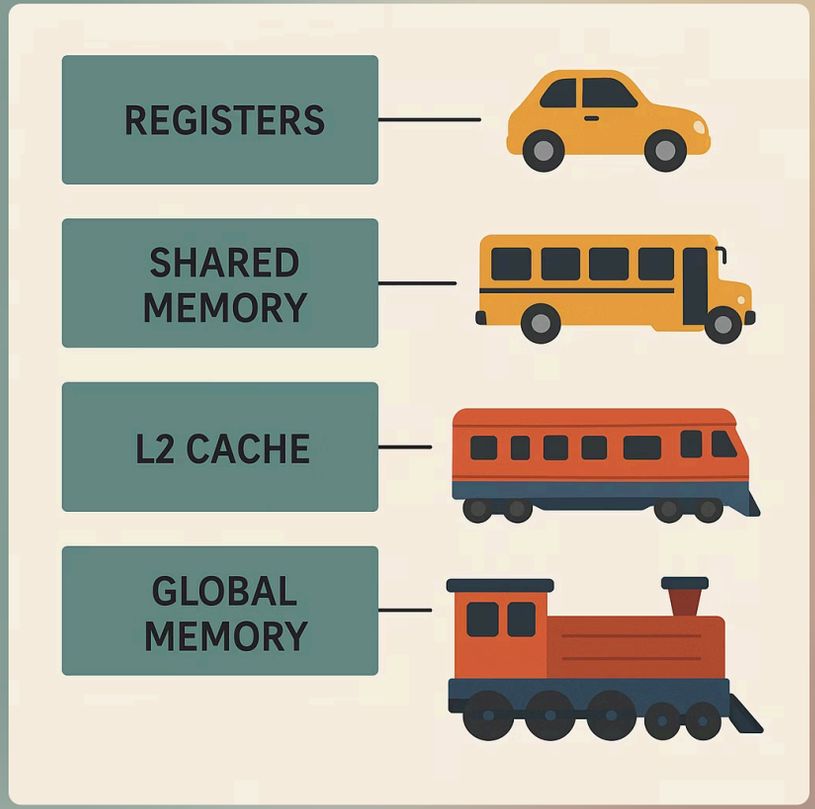
Batch Processing

Mini-batch gradient updates reduced overhead and allowed samples in one batch to be processed in parallel.

4

Tensor Core Usage

Replacing matrix multiplications with cuBLAS GEMM calls using tensor ops.



System Configuration and Key Algorithms

System Configuration

- CPU: Intel i7-12700H (14C/20T)
- GPU: RTX 3080 (10GB VRAM, 8704 CUDA cores)
- Memory: DDR4 16GB @ 3200MHz
- OS: Ubuntu 22.04 LTS
- CUDA: Toolkit 12.4, cuBLAS 12.4
- Driver: NVIDIA 550.144.03

Key Algorithms

- Forward propagation (matrix multiplication + activation)
- Softmax normalization
- Backward propagation (gradient computation)
- Weight updates using stochastic gradient descent (v1-v3.1, v4) and mini-batch gradient descent (v3.2)

Performance Analysis and Findings

V2 Slower

Due to poor memory access and high CPU-GPU communication overhead.

V3.1-V3.2 Speedup

1.30–1.33% speedup due to better memory usage and optimizing some kernel launches.

V3.2.1 Huge Leap

41.76x from batch processing, unified kernels, and optimal launch configuration and warp efficiency.

V4 Slower

Despite tensor cores, slower than V3.2.1 due to the lack of batch processing and the small matrix sizes.

Our performance analysis revealed that V3.2.1 achieved a significant speedup of 41.76x due to batch processing and unified kernels. V4, despite using tensor cores, was slower due to the absence of batch processing and small matrix sizes. Proper GPU utilization is crucial for optimal performance.

Conclusion and Future Work

Key Takeaways

Proper GPU utilization (V3.2.1) outperforms even tensor-core-based naive implementations. Batch processing is a critical factor in speedup. cuBLAS and tensor ops are powerful but need well-aligned workloads.

Future Work

Explore larger batch sizes with more epochs for higher accuracy. Optimize tensor core usage with larger matrix sizes. Investigate advanced debugging tools for CUDA kernels.

Conclusion

Demonstrated GPU utilization and batch processing for faster neural network training

Next Steps

Optimize tensor cores and explore debugging tools to enhance performance and accuracy