HPC PROJECT
REPORT

Abstract

Four implementations of a fully connected neural network for MNIST classification are shown. The implementations evolve in performance from a pure sequential CPU version to a GPU kernel accelerated by means of tensor cores. Version 1 (V1) is considered the CPU baseline. Version 2 (V2) is simply a naïve CUDA port of V1. Version 3 (V3) explores optimizations for launch, occupancy, communication, and memory hierarchy. Version 4 (V4) exploits NVIDIA tensor cores in mixed precision. Execution time, training time, and test accuracy are reported for all versions, with ~54× speedup over CPU (V1→V4) with a slight degradation in accuracy.

Introduction

Parallel hardware is much required for the deep learning workload. Here, we make a minimum two-layer (input 784, hidden 128, output 10) network to be optimized at every next level:

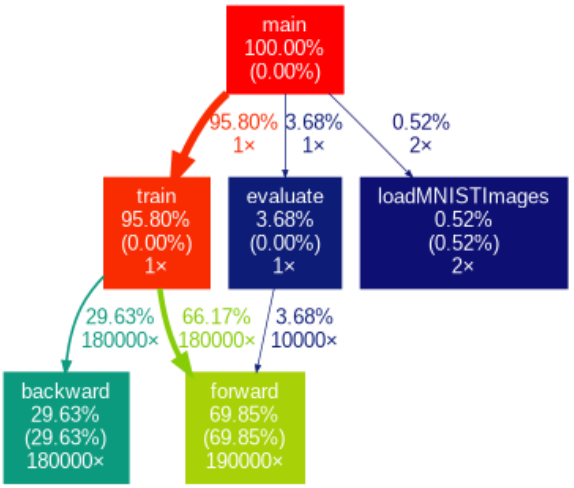
- V1: Sequential CPU coded in C.
- V2: Naive CUDA.
- V3: Optimized CUDA - launch, occupancy, communication and memory hierarchy.
- V4: Extend from V3 in exploiting tensor cores (FP16 inputs, WMMA API).

We evaluate all these versions in MNIST for total training time, end-to-end execution time, test loss, and accuracy.

V1: Sequential CPU Implementation

Results:
Total training time: 22.233 s
Test Accuracy: 96.78%.
End-to-end execution: 23.589 s.

Observations:
Loop-carried Dependencies (forward/backward).
No optimization - serial code.
Serves as the baseline for the speedup.



Profiling Insight: Forward/backward passes account for ~90% of runtime, dominated by the two nested loops

V2: Naïve CUDA

Key changes:
cudaMalloc/cudaMemcpy for data allocation/transfer on the GPU.
Two kernels: forward/backward pass - (based on profiling results from Version 1(V1)).

Results:
Total training time: 206.037 s
Test Accuracy: 96.58%
Execution time: 207.567 s

Observations:
GPU overhead (data transfers, kernel launches)
Low Resource Occupation - no optimization
Memory-bound operations are the bottleneck now - no communication optimizations

V3: Optimized CUDA Implementation

Optimizations applied:
Launch Configuration
    dim3 grids and blocks that optimize it for the matrix-vector computations.
Occupancy
    256 and 512 threads/block maximizing usage of SMs
Communication Optimization
    Batching of H2D/D2H copies with the asynchronous streams to allow for overlapping of copies with compute operations.
Memory hierarchy:
    Shared-memory tiling for dot products - reduce global memory access overhead
    Coalesced memory access

Results:
Total training time: 0.856s
Test Loss: 0.3440
Test Accuracy: 90.27%
Execution time: 1.097s

Observations:
~210 speedup in its training compared with V2.
The accuracy dropped by (probably due to differences in floating-point ordering).
Memory-bound operations are now well-optimized, with compute-bound kernels being the bottleneck now.

V4: Implementation Accelerated by Tensor Cores

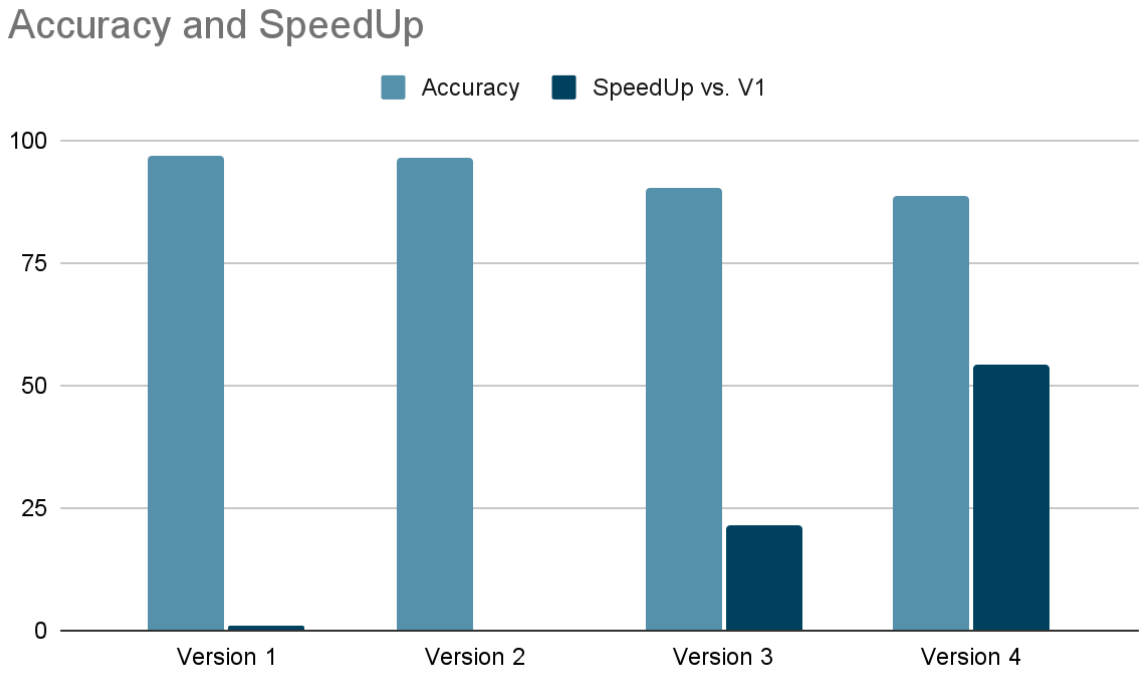
Approach:
Mixed precision
FP16(half) weights & & activations, FP32(float) accumulate.
NVIDIA WMMA API (warp-level matrix ops, 16×16 tiles).
Kept alignment and tile-multiples of 16 - maximizing tensor core occupnacy

Results:
Total training time: 0.262 s
Test loss: 0.4339
Test accuracy: 88.76%
Execution time: 0.435 s

Observations:
~3.3× faster than V3; ~54× faster than V1.
Some more accuracy drop seemed to be due to FP16 quantization and numeric dispersion.
Occupancy still high; tensor cores peak compute.

Comparative Performance

Version	Train Time (s)	Exec Time (s)	Test Acc. (%)	Speedup vs. V1
V1	22.233	23.589	96.78	1×
V2	206.037	207.567	96.58	0.11×
V3	0.856	1.097	90.27	21.5×
V4	0.262	0.435	88.76	54.2×



Discussion

Performance Gains:

It is inefficient to move from CPU to naive CUDA (V1 to V2), as doing so without algorithm changes results in small speedups because of overheads.

Optimizing launch, occupancy, communication and memory (V2 to V3) gives the biggest performance jump.

Tensor cores (V3 to V4) add further substantial benefits for loads that are mostly matrix based.

Accuracy Trade-offs:

The mixed-precision variant V4 also introduces accuracy loss which might be acceptable for many inference use cases while possibly implicating the need of retraining or loss-scaling.

Complexity Vs Benefit:

Coding complexity is added with V3 (shared-memory, stream management).

Understanding WMMA and precision management is needed in V4.

It increases the development effort and brings better returns through throughput.

Conclusion

The performance improvement trajectory showed a distinct drop from 23.6 s on CPU (V1) down to 0.44s using tensor cores (V4) at a speedup of ~54 ×. The accuracy drops slightly when one is too aggressive on optimizations - speedup.