Performance Analysis of GPU Optimization Strategies for Histogram Computation and Matrix Multiplication

Jaka Škerjanc, Erik Pahor

I. Introduction

This report presents an analysis of two GPU performance optimization tasks: histogram computation and matrix multiplication using TensorCores. The homework focuses on understanding the impact of different implementation strategies on GPU performance metrics and execution efficiency. The histogram computation task compares naive and privatized implementations, while the matrix multiplication task evaluates the benefits of TensorCore acceleration against traditional GPU computation methods.

II. Implementation Approach

Task 1: Histogram Computation

The implementation approach for the histogram computation involved two distinct strategies:

Naive Implementation

The naive implementation features:

- Direct updates to global memory
- Single histogram shared across all threads
- No privatization strategy

Optimized Implementation

The optimized version implements:

- Thread group privatization
- Local histogram copies per thread group
- Final merge step to combine private histograms

Task 2: Matrix Multiplication

The matrix multiplication implementation utilized two approaches:

Naive Implementation (mm_naive)

- Traditional GPU matrix multiplication
- Basic block-based approach
- Standard CUDA kernel implementation

TensorCore Implementation (mm_block_tc)

- WMMA PTX API utilization
- Block-based matrix multiplication
- TensorCore acceleration
- Configurable block sizes matching warp dimensions

III. Performance Results

A. Histogram Computation

The performance analysis was conducted with varying compute units (2, 4, and 8) and focused on the following metrics:

Table I
Histogram Performance Metrics

Metric	Naive			Optimized		
	8CU	4CU	2CU	8CU	4CU	2CU
Load Latency (cycles)	24.2M	12.4M	6.2M	4.1M	1.6M	0.8M
Vector ALU Insts	5.6K	11.1K	22.5K	7.2K	14.3K	28.7K
Shared Mem Reads	0	0	0	512	1.0K	2.0K
Shared Mem Writes	0	0	0	512	1.0K	2.0K
Bank Accesses	0	0	0	98.3K	196.6K	393.2K
Total Cycles	791.8K	802.9K	827.2K	174.5K	176.2K	227.6K
VPC	0.993	1.928	3.803	7.136	14.131	21.887

B. Matrix Multiplication

Performance comparison was conducted across different matrix sizes:

Table II
Matrix Multiplication Performance Comparison

Matrix Size	TensorCore (ms)	Without TensorCore (ms)	Speedup
256×256	0.016	0.224	14.38×
512×512	0.040	1.543	38.38×
1024×1024	0.327	11.724	35.88×
2048×2048	1.884	80.922	42.96×
4096×4096	15.329	881.823	57.53×

IV. Analysis

Task 1: Histogram Computation

The analysis reveals several key findings:

- Memory Contention
  - Higher memory contention in naive implementation
  - Reduced contention through privatization
- Performance Metrics
  - Improved load latency in optimized version
  - Better shared memory utilization
  - Reduced bank conflicts

Task 2: Matrix Multiplication

Key observations from the matrix multiplication analysis:

- Performance Scaling
  - Better scaling with larger matrices
  - Significant improvements for larger sizes
- Block Size Impact
  - Optimal block sizes matching warp dimensions
  - Trade-off between memory access and computation

V. Conclusion

The study demonstrates several key findings:

- Memory Access Optimization
  - Significant impact of proper memory access patterns
  - Effective reduction of contention through privatization
- Hardware Utilization
  - Substantial benefits from TensorCores
  - Importance of proper block sizing

These results demonstrate that GPU optimization strategies can significantly impact performance, with the choice of implementation approach depending on the specific workload characteristics and hardware capabilities.