Advanced Encryption Standard (AES) Implementation

CPU Reference and GPU Acceleration Study

Author: Subhranil Das
Date: 27 November 2025
Organisation: IITJ

Contributors
Name: Subhranil Das
Roll Number: G24AIT2010

Abstract

This report documents a comprehensive implementation and analysis of the Advanced Encryption Standard (AES) algorithm, with both CPU reference implementation and GPU acceleration using CUDA. The project focuses on AES in Electronic Codebook (ECB) mode with support for 128-bit, 192-bit, and 256-bit key sizes.

The implementation includes the following

- Complete CPU reference implementation of AES following FIPS-197 standard
- CUDS-accelerated GPU kernels for ECB mode encryption/decryption
- Comprehensive unit tests with known-answer vectors (KATs)
- Performance benchmarking comparing CPU and GPU implementations
- Detailed analysis of performance metrics across varying data sizes (1 KB to 100 MB)

Results demonstrate GPU acceleration provides 38x to 112x speed-up over CPU implementation, with GPU throughput reaching 1000 MB/s on smaller data sizes and maintaining 337.84 MB/s even at 100 MB data size. The project serves as both an educational resource for understanding AES algorithm details and a practical demonstration of GPU acceleration benefits for cryptographic operations.

Table of Contents

1. Introduction

1.1 Background

The Advanced Encryption Standard (AES) is a symmetric-key block cipher standardised by the National Institute of Standards and Technology (NIST) in 2001 (FIPS-197). It has become the de facto standard for cryptographic operations across the globe, replacing the older Data Encryption Standard (DES).

Key characteristics of AES:
- Symmetric-key algorithm (same key for encryption and decryption)
- Block cipher with fixed 128-bit blocks
- Support for three key sizes: 128-bit, 192-bit, and 256-bit
- 10, 12, or 14 rounds depending on key size (AES-128/192/256)
- Operates on a 4×4 byte state matrix using substitution-permutation network (SPN)

1.2 Motivation for GPU Acceleration

Cryptographic operations are computationally intensive and often represent a bottleneck in security-critical applications. Modern GPUs possess thousands of parallel processing cores capable of executing independent operations simultaneously, making them ideal for:

- Bulk encryption of large datasets
- Parallel processing of multiple independent blocks
- High-throughput cryptographic operations in data centres
- Real-time encryption in video streaming and secure communications

This project investigates the practical speed-up achievable by implementing AES encryption on NVIDIA GPUs using CUDA, compared to optimised CPU implementations.

1.3 Project Scope

This project focuses specifically on the following.

- AES ECB (Electronic Codebook) mode for simplicity and testability
- Support for AES-128, AES-192, and AES-256
- Performance benchmarking on data sizes from 1 KB to 100 MB

• Correctness validation through known-answer vectors and round-trip tests
• Comprehensive documentation and build infrastructure

Note: ECB mode is used for correctness testing and educational purposes only. It is not recommended for production use due to security limitations.

2. Literature Review

2.1 AES Algorithm Overview

The AES algorithm operates on 128-bit (16-byte) plaintext blocks and uses iterative rounds of substitution and permutation operations. The number of rounds varies by key size.

Following are the key sizes available for AES

• AES-128: 10 rounds
• AES-192: 12 rounds
• AES-256: 14 rounds

Following are the core transformations in each round.

1. Substitute Bytes: Non-linear substitution using 8×8 S-box lookup table
2. Shift Rows: Cyclic shift of rows in the state matrix
3. Mix Columns: Linear transformation in GF(2^8) using matrix multiplication
4. Add Round Key: XOR with round key material

The final round omits the Mix Columns step.

2.2 Galois Field Arithmetic

AES operations, particularly Mix Columns, rely on arithmetic in GF(2^8) (Galois Field with 256 elements). This finite field uses

• Irreducible polynomial: x^8 + x^4 + x^3 + x + 1 (0x1B in hex)
• Multiplication and addition defined modulo this polynomial

The Mix Columns operation multiplies each column of the state by a fixed matrix in GF(2^8):

[02 03 01 01]
[01 02 03 01]
[01 01 02 03]
[03 01 01 02]

This ensures diffusion of plaintext bits across the ciphertext.

2.3 GPU Acceleration Concepts

NVIDIA's CUDA framework enables GPU programming by

- Exposes GPU as a massively parallel co-processor.
- Uses thousands of lightweight threads organised in blocks/warps.
- Provides unified memory model and memory hierarchy (global, shared, local).
- Supports standard C/C++ with GPU-specific extensions.

Optimal GPU utilisation requires the following.

- Sufficient parallelism to occupy all cores.
- Minimised global memory bandwidth consumption.
- Maximised arithmetic intensity (compute-to-memory ratio).
- Coalesced memory access patterns.

For AES, ECB mode provides natural parallelism: each 16-byte block can be encrypted independently.

2.4 Related Work

Prior research on AES GPU acceleration has shown:

- 10-50x speed-up over CPU for bulk encryption in ECB/CBC modes.
- Trade-offs between memory usage and register pressure.
- Optimisation techniques: shared S-box caching, thread coalescing.
- Performance varies significantly with GPU architecture and data sizes.

This project implements a straightforward GPU kernel design prioritising correctness and clarity over aggressive optimisation.

3. Objectives

The primary objectives of this AES implementation project are the following.

1. Understanding
- Implement AES algorithm faithfully following FIPS-197 specification.
- Document all algorithm steps with clear code comments.
- Provide reference CPU implementation for comparison and validation.

2. GPU Acceleration Implementation
- Port AES operations to CUDA device kernels.
- Implement ECB mode kernels with one thread per block.
- Handle host-device memory transfers efficiently.

3. Correctness Validation
- Develop comprehensive test suite with known-answer vectors (FIPS-197).
- Support AES-128, AES-192, and AES-256 key sizes.

• Verify CPU and GPU implementations produce identical ciphertext.
• Implement multi-block round-trip tests.

4. Performance Analysis
• Benchmark CPU vs GPU performance across multiple data sizes.
• Measure encryption/decryption throughput (MB/s).
• Calculate GPU speed-up factors
• Analyse performance scaling characteristics.

5. Build Infrastructure
• Create portable Makefile-based build system.
• Support Windows (MSVC) and Linux (GCC) platforms.
• Separate test suite from main benchmarking code.
• Enable independent compilation of components.

4. Methodology / System Design

4.1 Project Architecture

The project follows a modular architecture separating concerns:

```
src/
├── aes_cpu.cpp - CPU reference implementation
├── aes_gpu.cu - GPU kernels and host wrappers
├── utils.cpp - Utility functions
├── main.cu - Performance benchmarking

include/
├── aes.h - Public API definitions
├── kernels.h - GPU kernel declarations
├── utils.h - Utility function declarations
└── aes_test.h - Test vector definitions

tests/
├── test_aes_ecb.cpp - CPU unit tests
├── test_aes_ecb_gpu.cu - GPU unit tests
└── main.cu - Unified test runner
```

Key design decisions:

• CPU and GPU implementations share common headers and type definitions.
• GPU implementation mirrors CPU algorithm for cross-validation.
• Separate compilation units for CPU and GPU code.
• Modular Makefile system with reusable source lists.

4.2 Build System Design

The build system uses GNU Make with modular configuration.

sources.mk - Defines source file lists (SRCS, CUDA_SRCS, HEADERS)
Makefile - Primary build orchestration

Build targets:
• all - Compile and link all sources.
• compile - Compilation only.
• run - Execute the benchmarking application.
• clean - Remove object files and executables.
• scrub - Remove entire build directory.

Platform detection:
• Windows: Uses .obj file extension, MSVC compiler via nvcc.
• Linux: Uses .o file extension, GCC compiler.

Separate test build:

• tests/Makefile manages independent test compilation.
• Tests can be built and run independently.
• Unified test runner aggregates CPU and GPU test results.

4.3 CPU Implementation Strategy

The CPU implementation provides the reference standard.

Key Features:

• Pure C++ implementation without external crypto libraries.
• Uses lookup tables for S-box and inverse S-box (256-element arrays).
• Implements GF(2^8) multiplication using precomputed Galois field operations.
• Provides functions for single-block and ECB multi-block encryption/decryption.
• Key expansion (key schedule) pre-computed before block processing.

Performance Characteristics:

• Optimised for clarity and correctness rather than performance.
• ~8.85 MB/s throughput on AES-128 ECB (single-threaded).
• Uses standard block cipher modes without special optimisations.

4.4 GPU Implementation Strategy

The GPU implementation parallelises ECB mode encryption:

Kernel Design:

• One CUDA thread processes one 16-byte block.
• Thread blocks contain multiple threads for better GPU utilisation.
• Each thread operates on independent block (natural parallelism).
• Shared lookup tables in device constant memory for S-box access.

Memory Organisation:
• Device constant memory: AES S-box and inverse S-box (512 bytes total).
• Global memory: Plaintext, ciphertext, and round keys.
• Thread local storage: State matrix (16 bytes per thread).

Kernel Structure:
1. Load block data from global memory into thread state.
2. Perform AES encryption rounds on state.
3. Store encrypted state back to global memory.

Host Wrapper Functions:
• Allocate GPU memory.
• Copy expanded keys and plaintext to device.
• Launch kernel with appropriate grid/block dimensions.
• Synchronise and copy results back to host memory.

4.5 Test Strategy

Comprehensive testing ensures correctness.

Unit Tests:

• Known-Answer Vector (KAT) testing with FIPS-197 vectors.
• Single-block encryption/decryption validation.
• Multi-block round-trip tests (encrypt then decrypt).
• Support for all three key sizes: 128, 192, 256-bit.

Test Coverage:

• AES-128 single-block known-answer test.
• AES-128/192/256 multi-block round-trip tests.
• CPU and GPU implementations tested independently.
• Cross-verification: CPU and GPU produce identical ciphertext.

Benchmark Validation:
• CPU/GPU results compared byte-by-byte for each data size.
• Immediate failure on mismatch prevents incorrect performance conclusions.
• Separate plaintext pattern for each run ensures independent data.

4.6 Benchmark Design

Performance evaluation uses realistic multi-block scenarios:

Benchmark Parameters:
• Test data sizes: 1 KB, 10 KB, 100 KB, 1 MB, 10 MB, 100 MB
• AES-128 ECB mode (primary focus)
• Plaintext pattern: sequential bytes (repeating 0-255 cycle)

Measurements:
• Elapsed time in milliseconds (std::chrono::high_resolution_clock)
• Throughput calculation: (data_size / duration) in MB/s
• Speed-up factor: CPU_time / GPU_time
• Correctness verification: memcmp(cpu_result, gpu_result)

Performance Metrics:
• Latency-bound behaviour at small sizes (1-10 KB)
• Memory bandwidth-bound behaviour at large sizes (100 MB)
• GPU overhead amortised across large transfers
• Realistic assessment of practical GPU benefits

5. Implementation

5.1 Core AES Algorithm

The AES encryption process follows FIPS-197 specification:

1. Key Schedule Generation (aes_expand_key)
• Expands initial key into round keys
• 10 round keys for AES-128 (11 total including initial)
• Uses RotWord, SubWord, Rcon lookup for key expansion
• Round key size: 16 bytes per round

2. Initial Round (AddRoundKey)
• XOR plaintext state with initial round key
• No substitution or permutation in initial step

3. Main Rounds (Nr-1 rounds)
• SubBytes: Apply S-box lookup to each state byte
• ShiftRows: Cyclic left shift of state rows
• MixColumns: Matrix multiplication in GF(2^8)
• AddRoundKey: XOR with current round key

4. Final Round (Nr round)
• SubBytes: Apply S-box
• ShiftRows: Cyclic shift
• AddRoundKey: XOR with final round key
• MixColumns is omitted in final round

Decryption reverses the process using inverse operations:
• InvSubBytes: Apply inverse S-box

• InvShiftRows: Cyclic right shift
• InvMixColumns: Inverse matrix operation in GF(2^8)

5.2 Galois Field Implementation

GF(2^8) operations are crucial for MixColumns:

CPU Implementation (aes_cpu.cpp):
• Uses lookup tables for multiplication (faster, larger code size)
• gmul(x, 2), gmul(x, 3), gmul(x, 9), gmul(x, 11), gmul(x, 13), gmul(x, 14)
• Pre-computed multiplication by constant factors

GPU Implementation (aes_gpu.cu):
• d_galois_mul() function implements field multiplication
• Bitwise iteration avoiding large lookup tables
• Balances memory constraints with acceptable performance
• __forceinline__ directive encourages compiler inlining

5.3 ECB Mode Implementation

Electronic Codebook mode enables parallel processing:

CPU ECB (aes_encrypt_ecb/aes_decrypt_ecb):
• Loops over plaintext/ciphertext in 16-byte blocks
• Calls single-block cipher for each block independently
• Sequential processing on CPU

GPU ECB (aes_encrypt_ecb_cuda/aes_decrypt_ecb_cuda):
• Maps each block to individual CUDA thread
• Kernel launches with grid_size = ceil(num_blocks / block_size)
• Each thread processes one independent block
• Natural parallelism across thousands of blocks

5.4 Memory Management

GPU memory operations require careful management:

Host Wrapper Functions:
1. Memory allocation
• cudaMalloc device memory for plaintext, ciphertext, round keys
• Check allocation errors via CUDA_CHECK macro

2. Data transfer (host → device)
• Copy plaintext via cudaMemcpy(..., cudaMemcpyHostToDevice)
• Copy expanded round keys via cudaMemcpy

3. Kernel execution
• Configure grid (number of blocks) and block (threads per block)

• Launch kernel: <<<grid, block>>> syntax
• Synchronize: cudaDeviceSynchronize()

4. Data transfer (device → host)
• Copy encrypted ciphertext via cudaMemcpy(..., cudaMemcpyDeviceToHost)

5. Cleanup
• Free device memory via cudaFree
• Handle error codes

5.5 Implementation Statistics

File	Lines	Purpose
aes_cpu.cpp	~1200	CPU AES implementation (all modes)
aes_gpu.cu	~800	GPU kernels and CUDA wrappers
main.cu	~150	Benchmarking harness
test_aes_ecb.cpp	~250	CPU unit tests
test_aes_ecb_gpu.cu	~200	GPU unit tests
Total	~2600	Complete project implementation

6. Results and Analysis

6.1 Performance Benchmark Results

Comprehensive benchmarking reveals GPU acceleration effectiveness across multiple data sizes:

Benchmark Results (AES-128 ECB):

Data Size	Number of Blocks	CPU Time (ms)	CPU (MB/s)	GPU Time (ms)	GPU (MB/s)	Speed-up	Match
1 KB	64	~0.11	~9091	~0.01	~100000	~11x	YES
10 KB	640	~1.1	~9091	~0.008	~1250000	~137x	YES
100 KB	6400	~11	~9091	~0.1	~1000000	~110x	YES
1 MB	65536	~115	~8870	~1.0	~1000000	~115x	YES
10 MB	655360	~1150	~8870	~10	~1000000	~115x	YES
100 MB	6553600	~11300	~8850	~296	~337840	~38x	YES

Key Observations:

1. CPU Performance (Baseline)
• Consistent throughput: ~8.85-8.87 MB/s
• Throughput independent of data size
• Single-threaded sequential processing

2. GPU Performance (CUDA Accelerated)
• Variable throughput dependent on data size
• Small data (1-10 KB): Limited by kernel launch overhead and latency
• Medium data (100 KB - 10 MB): Peak throughput ~1000 MB/s (113x CPU)
• Large data (100 MB): Throughput ~337.84 MB/s (38x CPU)

3. Speed-up Factors
• Small scale (1-10 KB): 11-137x due to parallel efficiency
• Medium scale (100 KB - 10 MB): 110-115x (peak efficiency)
• Large scale (100 MB): 38x (reduced by memory bandwidth saturation)

4. Correctness
• All CPU/GPU pairs produce identical ciphertext
• 100% verification pass rate across all data sizes

5. GPU Memory Bandwidth Limitations
• GPU peak bandwidth: ~200-300 GB/s (architecture dependent)
• At 100 MB: 337.84 MB/s represents bandwidth-limited regime
• Indicates GPU memory subsystem saturation at largest tested size

6.2 Test Suite Results

Unit Tests Pass All Validation Criteria:

CPU Tests (test_aes_ecb.cpp):

- AES-128 single-block known-answer test (FIPS-197 vector)
- AES-128 multi-block round-trip test (4 blocks = 64 bytes)
- AES-192 multi-block round-trip test
- AES-256 multi-block round-trip test

GPU Tests (test_aes_ecb_gpu.cu):

- AES-128 single-block GPU encryption/decryption
- AES-128 multi-block GPU round-trip test
- AES-192 multi-block GPU round-trip test
- AES-256 multi-block GPU round-trip test

Unified Test Runner (tests/main.cu):

- CPU and GPU tests execute sequentially

- Combined results: "All AES ECB CPU & GPU tests PASSED"
- Summary reports individual test outcomes

6.3 Performance Scaling Analysis

Performance scaling exhibits distinct regimes:

Regime 1: Latency Dominated (1-10 KB)
• GPU kernel launch overhead: ~0.5-1 ms
• Data transfer overhead significant relative to compute time
• Large speed-ups (11-137x) reflect overhead amortisation across small datasets

Regime 2: Bandwidth Limited - Peak Performance (100 KB - 10 MB)
• Kernel launch overhead negligible relative to compute time
• GPU cores well utilised with sufficient parallelism
• Consistent speed-up ~110-115x
• GPU throughput stable at ~1000 MB/s

Regime 3: Memory Saturation (100 MB)
• GPU memory bandwidth becomes bottleneck
• Global memory access patterns dominate execution time
• Speed-up drops to 38x as GPU throughput falls to 337.84 MB/s
• Further optimisation would require memory access patterns tuning

7. Discussion

7.1 GPU Acceleration Effectiveness

The GPU acceleration demonstrates substantial performance improvements:

Advantages of GPU Implementation:
• 38-115x speed-up provides significant practical benefit
• Peak throughput (1000 MB/s) far exceeds CPU capability (8.85 MB/s)
• Scalable to thousands of parallel blocks without code changes
• Demonstrates CUDA's effectiveness for data-parallel algorithms

Limitations and Trade-offs:
• Kernel launch overhead reduces efficiency on small data sizes
• Memory bandwidth becomes limiting factor at large scales
• GPU acceleration beneficial primarily for bulk encryption scenarios
• Not suitable for single-block operations due to overhead
• Requires GPU-compatible hardware and CUDA toolkit

7.2 Correctness and Validation

The implementation priorities correctness through multiple validation layers

Validation Approaches:
1. Known-Answer Vectors (KATs)
• Use FIPS-197 standard test vectors
• Verify correct S-box substitution and transformations
• Single-block validation against official specification

2. Cross-Implementation Validation
• CPU and GPU must produce identical results
• Byte-by-byte comparison after encryption
• Immediate failure on mismatch prevents invalid conclusions

3. Round-Trip Testing
• Encrypt : decrypt cycle must recover original plaintext
• Validates both forward and inverse operations
• Tests multiple key sizes and block counts

4. Multiple Key Sizes
• AES-128, AES-192, AES-256 all tested
• Verifies correct number of rounds (10, 12, 14 respectively)
• Key expansion logic validated indirectly

Validation Confidence:
• All tests pass 100% of the time
• No observed correctness issues across development
• Implementation suitable for educational and reference purposes

7.3 Implementation Quality

Code quality considerations reflect project goals:

Strengths:
• Clear separation between CPU and GPU implementations
• Comprehensive comments explaining algorithm steps
• Modular architecture enabling independent testing
• Error handling with CUDA_CHECK macro for debugging
• Cross-platform build support (Windows/Linux)

Areas for Future Enhancement:
• Aggressive optimisation for peak performance
• Support for additional cipher modes (CBC, CTR, GCM)
• Batch key expansion to amortise overhead
• Shared memory optimisation for S-box caching
• Register optimisation for state matrix storage

Current Design Philosophy:
• Prioritises correctness and clarity over performance
• Suitable for educational understanding and algorithm validation
• Provides solid foundation for further optimisation

7.4 Practical Applications

GPU-accelerated AES encryption has practical applications:

Suitable Use Cases:
• High-throughput encryption of large files/datasets
• Secure video streaming with real-time encryption
• Encrypted backups and data archival
• Network packet encryption at line-rate speeds
• Cryptographic protocols with bulk data encryption phase

Not Suitable:
• Single-block or very small dataset encryption (overhead dominates)
• Latency-critical applications (GPU transfer overhead ~1 ms)
• Resource-constrained environments without GPUs
• Scenarios requiring frequent CPU-GPU synchronisation

Performance Implications:
• 100 MB dataset: 11.3 seconds on CPU → 296 ms on GPU (38x speed-up = 10+ seconds saved)
• Practical benefit emerges quickly with moderate to large datasets
• Power efficiency considerations depend on GPU utilisation

8. Conclusion

This project successfully demonstrates GPU acceleration of the Advanced Encryption Standard (AES) algorithm with comprehensive validation and performance analysis.

Key Achievements:

1. Complete Implementation
• Faithful CPU reference implementation following FIPS-197
• GPU-accelerated ECB mode using CUDA
• Support for all three key sizes (128, 192, 256-bit)
• Production-ready correctness validation

2. Significant Performance Gains
• 38-115x GPU speed-up depending on data size
• Peak GPU throughput: 1000 MB/s vs CPU 8.85 MB/s
• Demonstrates practical effectiveness of GPU acceleration
• Performance scaling analysis identifies optimisation opportunities

3. Comprehensive Validation
• 100% test pass rate (CPU, GPU, and cross-verification)
• Known-answer vectors against official specification
• Round-trip testing validates forward/inverse operations
• Large-scale benchmarks confirm scalability

4. Educational Value

• Clear code documentation explaining algorithm steps
• Modular architecture enabling focused study
• Comparison framework (CPU vs GPU) for learning
• Practical demonstration of CUDA programming concepts

Limitations and Scope:
• Project uses ECB mode (not recommended for production)
• Focuses on throughput rather than latency optimisation
• Hardware-specific performance (tested on specific GPU)
• Provides foundation for further optimisation research

Overall Assessment:
The implementation successfully achieves its objectives, providing both an educational platform for understanding AES and a practical demonstration of GPU acceleration benefits. The 38-115x speed-up demonstrates that GPU acceleration is highly effective for bulk encryption operations, though with context-dependent overhead considerations. The project serves as a strong foundation for further research into cryptographic acceleration techniques.

9. Future Work

9.1 Algorithm Enhancements

1. Additional Cipher Modes
• CBC (Cipher Block Chaining) with IV dependency
• CTR (Counter) mode for stream cipher-like behaviour
• GCM (Galois/Counter Mode) for authenticated encryption
• CFB (Cipher Feedback) for streaming data

2. Key Management
• Batch key expansion for multiple keys simultaneously
• Key derivation functions (KDF)
• Key schedule caching for repeated encryption with same key

3. Performance Optimisation
• Shared memory caching of S-box for better locality
• Warp-level primitives for better GPU utilisation
• Register optimisation for state matrix representation
• Coalesced memory access patterns

9.2 Extended Functionality

1. Additional AES Variants
• Support for XTS mode (disk encryption)
• EAX mode (authenticated encryption)

• Support for hardware AES-NI instructions on CPU

2. Integration Capabilities
• Library interface for integration into larger systems
• OpenSSL/BoringSSL compatibility layer
• Python/Java bindings for broader accessibility

3. Security Features
• Constant-time implementation to prevent timing attacks
• Protection against side-channel attacks
• Memory clearing after sensitive operations
• Secure random IV generation

9.3 Measurement and Analysis

1. Extended Benchmarking
• Performance across different GPU architectures
• Power consumption analysis (performance per watt)
• Latency metrics for streaming applications
• Cache behaviour and memory traffic analysis

2. Comparative Studies
• Performance vs other GPU crypto libraries
• CPU-GPU load balancing strategies
• Hybrid encryption for varied data sizes
• Scaling across multiple GPUs

3. Security Analysis
• Formal verification of constant-time properties
• Side-channel vulnerability assessment
• Implementation against known attacks

10. References

[1] National Institute of Standards and Technology (NIST). Federal Information Processing Standards Publication 197: Specification for the Advanced Encryption Standard (AES). 2001. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf
[2] Joan Daemen and Vincent Rijmen. The Design of Rijndael: AES – The Advanced Encryption Standard. Springer-Verlag, 2002.
[3] NVIDIA Corporation. CUDA C++ Programming Guide. 2024. https://docs.nvidia.com/cuda/cuda-c-programming-guide/
[4] NVIDIA Corporation. Compute Capability Reference. 2024. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#compute-capabilities
[5] David Patterson and John Hennessy. Computer Architecture: A Quantitative Approach (6th Edition). Morgan Kaufmann, 2017.

[6] William Stallings. Cryptography and Network Security: Principles and Practice (7th Edition). Pearson, 2017.
[7] Agner Fog. Software Optimization Resources. 2024. https://www.agner.org/optimize/
[8] Cedric Nugteren. CLBlast: A Tuned OpenCL BLAS Library. 2024.
[9] Barry Marshall et al. Accelerating AES on Modern Multicore Processors. In AFRICACRYPT, 2016.
[10] Krystian Matusiewicz et al. Fast and Flexible Parallel AES Encryption/Decryption on GPUs. In SECRYPT, 2012.
[11] Shuai Che et al. A Characterization of Scalable Database Workloads on GPUs. Journal of Parallel and Distributed Computing, 2013.
[12] Jens Krüger and Rüdiger Westermann. Linear Algebra Operators for GPU Implementation of Numerical Algorithms. ACM Transactions on Graphics, 2003.

11. Appendix

11.1 Build Instructions

Quick Start (from CODE directory):

cd AES/CODE
`make clean`
`make all`

This compiles and runs the AES-128 ECB benchmarking application.

Build Targets:
`make compile` - Compile sources only
`make run` - Execute compiled benchmark
`make clean` - Remove object files
`make scrub` - Remove entire build directory

Test Suite (from CODE/tests directory):

cd AES/CODE/tests
`make clean`
`make all`
`make run`

This compiles and runs CPU and GPU unit tests.

Windows Considerations:
• Requires Visual Studio Build Tools for MSVC compiler
• CUDA toolkit must be installed with GPU support
• Object files use '.obj' extension (auto-detected by build system)

Linux Considerations:
• GCC compiler required
• CUDA toolkit installation required
• Object files use ‘.o’ extension

11.2 FIPS-197 Test Vector (AES-128)

FIPS Publication 197 specifies test vectors for AES validation.

Key (128-bit / 16 bytes):
2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

Plaintext (128-bit / 16 bytes):
32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

Expected Ciphertext (128-bit / 16 bytes):
39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32

This vector is used in aes_test_ecb_key128_singleblock() to validate correct AES-128 ECB operation.

11.3 GPU Architecture Details

NVIDIA GPU Architecture (General Principles):

Thread Organisation:
• Threads organised in blocks (1D, 2D, or 3D)
• Blocks organised in grid (1D, 2D, or 3D)
• Maximum threads per block: 1024 (typical)
• Maximum thread dimensions vary by architecture

Memory Hierarchy:
• Registers: Per-thread local fast memory
• Shared Memory: Per-block fast shared storage (~96 KB typical)
• Global Memory: Device-wide large memory (~GB size)
• Constant Memory: Read-only cached data (64 KB)

Execution Model:
• Warps: 32 threads execute in lockstep
• Warp Occupancy: GPU utilisation metric
• Latency Hiding: Many warps hide memory latency

For AES ECB Implementation:
• Block size: 256-512 threads typical

• Each thread processes one 16-byte block
• Constant memory stores AES S-box (512 bytes)
• Global memory patterns optimised for coalescing

11.4 Code Statistics

Project Source Code Breakdown:

aes_cpu.cpp (1173 lines):
• Core AES algorithm implementation
• S-box and inverse S-box tables (256 entries each)
• Galois field multiplication lookup tables
• Key expansion logic
• Single-block cipher functions
• ECB multi-block wrappers
• Additional modes: CBC, CTR (not used in this project)

aes_gpu.cu (783 lines):
• Device constant memory S-boxes
• Device transformation helper functions
• Galois field multiplication (bitwise)
• AES cipher kernels (encrypt/decrypt)
• Host wrapper functions with CUDA memory management
• Error checking and synchronisation

main.cu (150 lines):
• Benchmark harness
• Memory allocation and initialisation
• Timing measurements using std::chrono
• CPU/GPU execution and result comparison
• Throughput and speed-up calculations
• 6-tier benchmark (1 KB to 100 MB)

test_aes_ecb.cpp (249 lines):
• 4 CPU test functions
• Known-answer vector tests
• Multi-block round-trip tests
• Test result formatting with [CPU] prefix

test_aes_ecb_gpu.cu (200 lines):
• 4 GPU test functions mirroring CPU tests
• CUDA wrapper for GPU execution
• Result comparison and verification
• Test result formatting with [GPU] prefix

Total Implementation: ~2600 lines of code/tests.

11.5 Performance Calculation Examples

Performance Metric Calculations:

Example: 100 MB Benchmark

CPU Execution:
Time: 11,300 milliseconds
Data: 104,857,600 bytes = 100 MB
Throughput = 100 MB / (11,300 ms / 1000) = 8.85 MB/s

GPU Execution:
Time: 296 milliseconds
Data: 104,857,600 bytes = 100 MB
Throughput = 100 MB / (296 ms / 1000) = 337.84 MB/s

Speed-up:
Speed-up = CPU_time / GPU_time = 11,300 ms / 296 ms = 38.18x

Performance Efficiency:
GPU Efficiency = GPU_throughput / GPU_peak_throughput
Assuming GPU peak ~1000 MB/s: 337.84 / 1000 = 33.8% efficiency

Memory Bandwidth:
Effective bandwidth = throughput × bytes_per_element
For AES: throughput directly indicates device memory bandwidth usage

Smaller Example: 10 MB Benchmark

CPU Execution:
Time: ~1150 milliseconds
Throughput = 10 MB / 1.150 s = 8.70 MB/s

GPU Execution:
Time: ~10 milliseconds
Throughput = 10 MB / 0.010 s = 1000 MB/s

Speed-up:
Speed-up = 1150 ms / 10 ms = 115x

11.6 Command Reference

Build System Commands:

Main Application (CODE/):
`make all` - Build and run benchmark
`make compile` - Compile only
`make run` - Run compiled executable
`make clean` - Remove build artifacts
`make scrub` - Remove entire build/ directory

Test Suite (CODE/tests/):
`make all` - Compile and run tests
`make compile` - Compile test sources
`make run` - Run test executable
`make clean` - Remove test build artifacts

Direct Executable Execution:
`./build/out.exe` - Run benchmark (Windows)
`./build/out` - Run benchmark (Linux)
`./build/tests_out.exe` - Run tests (Windows)
`./build/tests_out` - Run tests (Linux)

Expected Output:
Benchmark: Shows 6 benchmark results (1 KB to 100 MB)
Each shows CPU/GPU timing, throughput, speed-up

Tests: Shows all test results with [CPU] and [GPU] prefixes
Final line: "All AES ECB CPU & GPU tests PASSED"