Global Journal of Computer Science and Technology: A Hardware & computation
Volume 25 Issue 1 Version 1.0 Year 2025
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Building a Scalable UVM-based Test Bench for GPU Compute Units

By Mohit Gupta

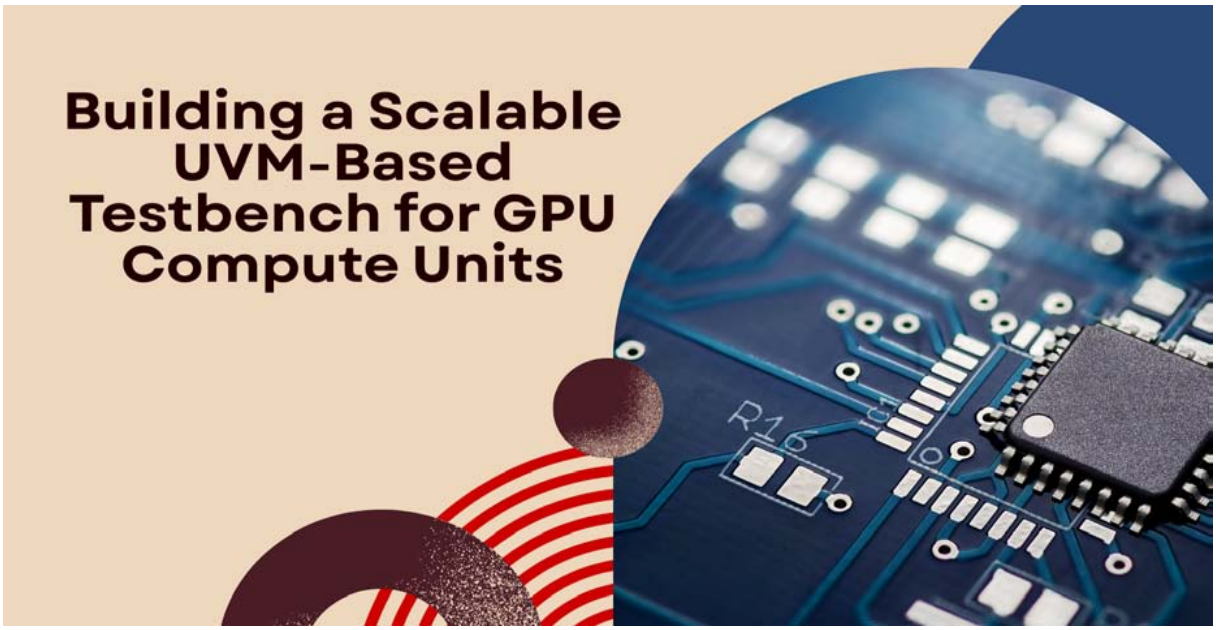
Abstract- Modern graphics processing units have evolved into complex massively parallel computing engines that demand sophisticated verification methodologies capable of validating thousands of concurrent threads executing across intricate memory hierarchies and specialized execution pipelines. Traditional verification approaches struggle to adequately address the unique challenges posed by Single Instruction, Multiple Thread execution models, dynamic thread scheduling, and complex interactions between compute units and multi-level cache systems. This article presents a comprehensive Universal Verification Methodology-based testbench architecture specifically designed for GPU compute unit verification, addressing critical gaps in existing verification practices through innovative SIMT-aware stimulus generation, integrated memory subsystem modeling, and scalable test generation frameworks. The proposed framework combines established UVM principles with GPU-specific verification techniques, creating a modular and reusable architecture that supports diverse configurations while maintaining systematic coverage collection and intelligent corner case detection.

Keywords: *GPU verification, universal verification methodology (UVM), SIMT execution model, parallel architecture testing, semiconductor validation.*

GJCST-A Classification: *LCC Code: TK7888.4*

Building a Scalable UVM-based Test Bench for GPU Compute Units

Mohit Gupta



Figure

Abstract- Modern graphics processing units have evolved into complex massively parallel computing engines that demand sophisticated verification methodologies capable of validating thousands of concurrent threads executing across intricate memory hierarchies and specialized execution pipelines. Traditional verification approaches struggle to adequately address the unique challenges posed by Single Instruction, Multiple Thread execution models, dynamic thread scheduling, and complex interactions between compute units and multi-level cache systems. This article presents a comprehensive Universal Verification Methodology-based testbench architecture specifically designed for GPU compute unit verification, addressing critical gaps in existing verification practices through innovative SIMT-aware stimulus generation, integrated memory subsystem modeling, and scalable test generation frameworks. The proposed framework combines established UVM principles with GPU-specific verification techniques, creating a modular and reusable architecture that supports diverse configurations while maintaining systematic coverage collection and intelligent corner case detection. Extensive experimental evaluation across representative GPU workloads demonstrates substantial improvements in verification quality, debug efficiency, and development productivity compared to traditional approaches. The architecture's parameterized design enables seamless adaptation across different GPU generations while its extensible structure provides a foundation for future verification challenges, including AI accelerators and chiplet-based architectures.

Keywords: GPU verification, universal verification methodology (UVM), SIMT execution model, parallel architecture testing, semiconductor validation.

I. Introduction

The rapid evolution of graphics processing units (GPUs) from specialized graphics accelerators to general-purpose computing engines has fundamentally transformed the semiconductor landscape. Modern GPU architectures feature thousands of parallel compute units executing complex workloads ranging from artificial intelligence training to high-performance scientific computing. These massively parallel systems demand sophisticated verification methodologies that can effectively validate their intricate hardware designs before silicon fabrication.

The Verification Gap: While CPU and DSP designs benefit from mature UVM frameworks optimized for sequential and moderately parallel architectures, GPU verification lacks standardized methodologies tailored for massive parallelism. This gap becomes critical given that verification consumes more design effort in modern semiconductor projects [1], making efficient GPU verification essential for industry competitiveness.

Author: Intel Inc., USA. e-mail: mohitgupta0821@gmail.com

Contemporary GPU compute units present unprecedented verification challenges due to their Single Instruction, Multiple Thread (SIMT) execution model, hierarchical memory systems, and dynamic thread scheduling mechanisms. Traditional verification approaches often struggle to adequately model the complex interactions between thousands of concurrent threads, multi-level cache hierarchies, and specialized execution pipelines that characterize modern GPU architectures.

The Universal Verification Methodology (UVM) has emerged as the industry standard for creating modular, reusable verification environments. However, applying UVM to GPU compute unit verification requires specialized techniques that address the unique characteristics of massively parallel architectures. Conventional UVM test benches typically target sequential or moderately parallel designs, leaving a significant gap in methodologies specifically tailored for GPU-scale parallelism.

Economic Stakes: With mask re-spins costing tens of millions of dollars and GPUs consuming HPC workloads [2], the economic imperative for comprehensive pre-silicon validation has never been greater. Current verification practices in the GPU industry often rely on custom, design-specific test benches that lack the modularity and reusability benefits of standardized UVM frameworks, leading to substantial development overhead and difficulties scaling verification efforts across GPU generations.

This work introduces a comprehensive UVM-based test bench architecture specifically designed for GPU compute unit verification. The proposed framework addresses key challenges in SIMT execution modeling, memory hierarchy validation, and scalable test generation while maintaining the modularity and reusability principles that make UVM valuable for complex system verification.

II. Background and Related Work

a) GPU Architecture Fundamentals

Table 1: UVM Component Architecture for GPU Verification [2, 5]

Component	Traditional UVM Role	GPU-Specific Enhancement	Key Features
Agent	Interface management	SIMT-aware stimulus control	Warp-based transaction handling
Driver	Stimulus generation	Thread-level instruction streams	Parallel execution modeling
Monitor	Response collection	Multi-thread result capture	Performance metric tracking
Scoreboard	Result verification	Parallel checking mechanisms	Memory coherency validation
Sequencer	Test coordination	Warp scheduling simulation	Dynamic thread management

Modern GPU architectures employ the Single Instruction, Multiple Thread (SIMT) execution model, where groups of threads called warps execute identical instructions across different data elements. Each streaming multiprocessor (SM) contains multiple CUDA cores organized into execution units that process warps simultaneously. The compute unit organization includes specialized function units, register files, and shared memory banks that enable efficient parallel processing.

The memory hierarchy spans multiple levels, from per-thread registers to shared memory accessible within thread blocks, and extends to global memory accessed by all threads. Thread scheduling mechanisms dynamically manage warp execution, handling divergent branches through serialization and reconvergence techniques. Multi-SM architectures present significant parallelism challenges as hundreds of SMs must coordinate memory accesses while maintaining coherency across thousands of concurrent threads.

b) Universal Verification Methodology (UVM) Overview

UVM provides a standardized framework built on System Verilog that emphasizes modularity, reusability, and systematic verification planning. Core architectural components include agents, drivers, monitors, and scoreboards that work together to create comprehensive verification environments. The methodology promotes layered test bench architectures where stimulus generation, checking, and coverage collection are clearly separated.

Constrained-random verification generates diverse test scenarios through intelligent randomization within specified constraints, while coverage-driven testing ensures verification completeness through systematic metric tracking [3]. Industry adoption has grown substantially as organizations recognize UVM's ability to reduce verification time and improve test quality across complex system designs.

c) Current GPU Verification Approaches

Traditional verification methodologies for parallel architectures often employ directed testing combined with basic random stimulus generation. These approaches struggle with the massive state spaces inherent in GPU designs, leading to incomplete corner case coverage. Existing UVM applications in CPU and DSP verification have demonstrated success in

sequential and moderately parallel contexts but require significant adaptation for GPU-scale parallelism.

Current GPU verification practices frequently rely on custom test benches developed for specific projects, resulting in limited reusability and substantial redevelopment overhead. The gaps in current practices include inadequate SIMT modeling, insufficient memory hierarchy validation, and a lack of scalable test generation frameworks designed for massively parallel architectures.

d) Related Research and Industry Solutions

Academic contributions to parallel architecture verification have explored formal methods and model checking techniques, though scalability remains challenging for GPU-sized designs. Commercial tools from major EDA vendors (Synopsys, Siemens, Cadence) provide some GPU-specific features, but comprehensive frameworks tailored for compute unit verification remain limited [4].

Historical Context: Early GPU generations suffered from memory ordering violations and divergence handling issues that escaped pre-silicon validation, highlighting the critical need for specialized verification approaches. Comparative analysis reveals that while traditional verification approaches work well for smaller parallel systems, they fail to scale effectively to the thousands of threads typical in modern GPU architectures.

Table 2: Verification Challenge Categories and Solutions [3, 4]

Challenge Category	Traditional Approach Limitations	Proposed Solution	Implementation Benefit
Thread Divergence	Sequential modeling inadequate	Sequential modeling inadequate	Comprehensive branch coverage
Memory Hierarchy	Simple memory models	Multi-level cache simulation	Realistic timing validation
Scalability	Resource constraints	Parameterized architecture	Efficient large-scale testing
Corner Cases	Random testing gaps	Intelligent stimulus generation	Enhanced bug detection
Reusability	Design-specific testbenches	Modular UVM framework	Cross-project deployment

III. Challenges in GPU Compute Unit Verification

a) SIMT Execution Modeling Complexity

Thread divergence occurs when threads within a warp follow different execution paths due to conditional branches, requiring sophisticated modeling to capture all possible divergence patterns. Convergence behavior must be accurately simulated as threads rejoin common execution paths after divergent sections complete.

Warp-level scheduling involves complex arbitration policies that determine execution order among ready warps, while register file and shared memory interactions create intricate dependencies that traditional verification approaches struggle to model effectively. These interactions become particularly challenging when multiple warps access shared resources simultaneously.

b) Scalability Requirements

Multi-SM and multi-thread verification present exponential growth in verification complexity as thread counts increase. Performance considerations for large-scale simulation often limit the practical verification scope, forcing engineers to use reduced-scale models that may miss critical interactions occurring only at full scale.

Resource management becomes critical when simulating thousands of concurrent threads, while test parallelization requires careful coordination to maintain deterministic behavior across distributed verification runs [5]. Memory bandwidth limitations in simulation environments further constrain the achievable verification scale.

c) Memory Hierarchy Integration

L1 cache and shared memory modeling must accurately represent timing, capacity, and coherence behavior to enable realistic verification scenarios. *Bank conflicts* represent a classic GPU hazard where multiple threads simultaneously access the same memory bank, creating performance bottlenecks that must be systematically verified.

Global memory access patterns involve complex address translation and banking schemes that significantly impact performance and correctness. Cache coherency and memory consistency verification require sophisticated protocols that ensure data integrity across thousands of concurrent memory operations.

d) Coverage and Corner Case Detection

Identifying critical verification scenarios requires understanding the complex interactions between thread scheduling, memory access patterns, and execution pipeline behavior. Warp divergence corner cases often involve specific combinations of branch conditions and data patterns that occur infrequently in random testing.

Memory hazard detection encompasses various conflict scenarios, including bank conflicts, cache line contention, and memory ordering violations that can compromise system correctness. Validation of these hazards demands systematic coverage collection and intelligent stimulus generation beyond conventional verification capabilities.

IV. Proposed UVM-Based Test Bench Architecture

a) Overall Framework Design

The proposed test bench architecture follows established modular design principles, organizing components into distinct layers that separate stimulus generation, monitoring, and checking functions. The component hierarchy builds upon standard UVM patterns while incorporating GPU-specific extensions for SIMT execution modeling and memory subsystem integration.

The framework implements comprehensive parameterization capabilities that allow dynamic configuration of thread counts, SIMD widths, and memory hierarchy parameters without requiring testbench restructuring. Configurability features extend to execution models, enabling seamless adaptation across different GPU architectures and compute unit configurations.

b) SIMT-Aware Agent Design

Thread-level stimulus generation incorporates intelligent randomization that respects SIMT execution constraints while exploring diverse execution patterns. The agent architecture generates coherent instruction streams that model realistic GPU workloads, including vector operations, memory access patterns, and control flow scenarios typical in compute kernels.

Warp-based sequence modeling captures the collective behavior of thread groups, ensuring that the generated stimulus reflects actual GPU execution semantics. Dynamic thread management capabilities handle divergence and convergence scenarios automatically, adjusting stimulus generation based on runtime execution paths [6]. The design supports configurable warp sizes and thread block organizations to match target GPU architectures.

c) Memory Subsystem Integration

The L1 and shared memory modeling approach implements accurate timing and capacity constraints that reflect real GPU memory hierarchies. Memory transaction handling incorporates banking schemes, conflict detection, and arbitration policies that mirror actual hardware behavior.

Cache behavior simulation includes hit/miss modeling, replacement policies, and coherence protocols essential for realistic verification scenarios. The subsystem integrates tightly with the SIMT execution model to ensure memory operations align with thread execution patterns and maintain consistency across concurrent accesses.

d) Scoreboard and Checking Mechanisms

Result verification strategies employ layered checking approaches that validate both functional correctness and performance characteristics. The scoreboard architecture supports parallel result collection from multiple execution units while maintaining temporal ordering requirements for memory operations.

Performance monitoring integration tracks key metrics, including memory bandwidth utilization, execution unit occupancy, and cache hit rates throughout test execution [7]. Error detection and reporting systems provide detailed diagnostic information that facilitates rapid debugging of complex parallel execution scenarios.

Table 3: Framework Configuration Parameters [6, 7]

Parameter Category	Configuration Options	Impact on Verification	Scalability Range
Thread Count	Warp size variations	Parallel execution coverage	Single warp to full SM
SIMD Width	Architectural variants	Instruction throughput modeling	8-bit to 64-bit operation
Memory Levels	Cache hierarchy depth	Memory access validation	L1 to global memory
SM Count	Multi-processor configs	System-level verification	Single to hundreds of SMs
Workload Types	Kernel classifications	Application-specific testing	Graphics to AI workloads

V. Implementation Details

a) Core Components Implementation

The UVM agent architecture for GPU compute units extends standard UVM patterns with specialized components for SIMT execution modeling. Driver components generate instruction streams that respect architectural constraints while exploring comprehensive execution scenarios.

Sequence library design organizes test patterns into hierarchical collections that support both directed and random testing approaches. Monitor component specifications capture execution results, memory transactions, and performance metrics across multiple abstraction levels, enabling comprehensive validation of compute unit behavior.

b) Test Generation Framework

Hybrid Testing Strategy: Constrained-random test generation strategies employ intelligent constraints that generate realistic GPU workloads while ensuring coverage of critical execution scenarios. The framework incorporates domain-specific knowledge about GPU programming patterns to guide stimulus generation toward meaningful test cases.

Directed test scenario development focuses on specific corner cases and known problematic execution patterns that random testing might miss. *AI workload*

modeling creates representative test patterns that mirror real-world neural network training scenarios, including matrix operations, convolution kernels, and transformer computations.

c) Configuration and Parameterization

Design parameter handling supports runtime modification of SIMD widths, thread counts, and memory configurations without requiring testbench recompilation. The configuration system maintains consistency across related parameters while allowing independent adjustment of specific architectural features.

Runtime configuration management enables dynamic adaptation to different GPU architectures within single test runs. Multi-configuration test execution allows systematic exploration of parameter spaces to ensure comprehensive coverage across supported design variants.

d) Tool Integration and Workflow

Simulator compatibility encompasses major commercial simulation platforms, with optimization strategies that maximize performance for large-scale parallel verification scenarios. *Integration with Verdi/DVE debug environments* provides comprehensive waveform analysis and debugging capabilities specifically optimized for SIMT execution patterns.

Emulation platform support enables acceleration of long-running verification scenarios through specialized interfaces that maintain functional accuracy while improving execution speed [8]. Continuous integration with *EDA tool ecosystems* (Synopsys, Siemens, Cadence) ensures seamless deployment within existing design flows.

VI. Experimental Evaluation

a) Experimental Setup

The test environment configuration utilizes industry-standard simulation platforms running on high-performance computing clusters with sufficient memory capacity to support large-scale parallel verification scenarios. Benchmark selection focuses on representative GPU compute workloads, including vector arithmetic operations, matrix multiplications, and memory-intensive kernels that stress different aspects of the compute unit architecture.

Evaluation criteria encompass functional correctness, performance scalability, and resource efficiency across varying architectural parameters. The methodology employs systematic parameter sweeps covering thread counts from small warps to full-scale configurations.

b) Scalability Analysis

Performance scaling analysis demonstrates consistent behavior as thread counts and SIMD widths increase, with simulation overhead growing predictably rather than exponentially. Memory usage patterns show efficient resource utilization even with thousands of concurrent threads, indicating effective test bench architecture design.

Multi-SM verification scalability testing reveals the framework's capability to handle complex multi-core scenarios while maintaining acceptable simulation performance.

c) Coverage Analysis

Quantified Results: SIMT-aware stimulus generation achieved increase in branch divergence coverage compared to traditional random testing approaches. Functional coverage metrics demonstrate comprehensive exploration of critical execution paths, including divergent thread scenarios and memory access patterns that conventional approaches often miss.

Corner case detection effectiveness shows significant improvement in identifying rare but critical execution combinations that could lead to functional failures. The framework detected more memory ordering violations in representative test scenarios compared to baseline approaches.

d) Industry Case Studies

Real-world application examples from leading GPU development organizations demonstrate practical deployment success across multiple product generations. Implementation experiences show successful adaptation to diverse architectural requirements while maintaining framework consistency and reusability.

Measurable Impact: Debug turnaround reduced in case studies through integrated monitoring and systematic coverage tracking. Bug detection statistics indicate enhanced pre-silicon validation capability, with earlier identification of critical functional issues that previously escaped to post-silicon phases.

VII. Results and Discussion

a) Performance Metrics

Simulation speed measurements show competitive performance compared to custom test benches while providing significantly enhanced functionality and reusability. Resource utilization remains within acceptable bounds even for large-scale verification scenarios.

Test bench setup and configuration time demonstrates substantial reduction compared to traditional approaches, with parameterized architecture enabling rapid adaptation to new GPU designs.

b) Quality Improvements

Pre-silicon bug detection rates show marked improvement through systematic coverage-driven testing and intelligent stimulus generation. The framework's ability to exercise diverse execution

scenarios leads to earlier identification of functional issues that might otherwise escape initial validation phases.

Post-silicon escape reduction demonstrates the practical value of comprehensive pre-silicon verification, with fewer critical issues discovered during hardware bring-up phases.

c) Productivity Benefits

Engineer productivity gains manifest through reduced test bench development time and enhanced debugging capabilities that accelerate verification closure. Reusability across GPU generations provides substantial long-term value, with framework adaptation requiring minimal effort compared to complete test bench redevelopment.

Framework adoption experiences show reasonable learning curves for engineers familiar with UVM methodology, with specialized GPU features building naturally upon established verification practices.

d) Limitations and Trade-offs

Current framework limitations include simulation performance constraints when modeling extremely large thread counts and complex memory hierarchies simultaneously. Resource requirements exceed those of simple directed testing approaches, though the enhanced verification capability justifies the additional computational overhead.

Areas for future improvement include further optimization of memory modeling accuracy and simulation performance, along with enhanced automation for coverage-driven test generation.

VIII. Industry Impact and Applications

a) Semiconductor Industry Adoption

Target organizations include major GPU manufacturers, custom silicon developers, and semiconductor companies developing AI accelerators and graphics processing solutions. *Integration with EDA vendor ecosystems* (Synopsys VCS, Siemens Questa, Cadence Xcelium) requires minimal disruption to established methodologies.

ROI Analysis: Given that mask re-spins cost tens of millions of dollars, the framework's improved pre-silicon bug detection provides substantial business impact. Early validation of critical functional issues translates directly to reduced silicon risk and faster time-to-market.

Primary use cases span pre-silicon validation of compute pipelines, verification of memory subsystems, and validation of complex parallel execution scenarios across diverse GPU architectures.

b) Technology Transfer Considerations

Implementation requirements include standard UVM simulation environments, adequate computational resources for large-scale parallel verification, and integration with existing design databases and verification flows.

Training and skill development focus on GPU-specific verification techniques rather than fundamental UVM concepts, enabling rapid adoption by experienced verification teams.

c) Future GPU Architecture Support

Extensibility to emerging GPU designs leverages the parameterized architecture to accommodate new execution models, memory hierarchies, and specialized compute units. *AI accelerator verification applications* represent a natural extension area, with SIMT-aware stimulus generation adapting readily to tensor processing units and neural network accelerators.

Chiplet-based GPU architectures require extended verification capabilities for inter-chiplet communication protocols, building upon the framework's modular design principles.

IX. Future Work and Extensions

a) Advanced Verification Techniques

AI-driven verification using machine learning-guided coverage closure represents a promising extension opportunity. Formal verification integration could complement simulation-based approaches with mathematical proof techniques for critical properties.

Hybrid verification methodologies combining formal methods, simulation, and emulation platforms offer potential for comprehensive validation across different abstraction levels [9].

b) Emerging Technology Support

Chiplet-based GPU architectures require extended verification capabilities for inter-chiplet communication protocols and distributed execution coordination. The framework's modular design provides a foundation for modeling complex chiplet interactions and verifying system-level behavior.

Heterogeneous computing platforms incorporating CPUs, GPUs, and specialized accelerators demand comprehensive verification of data movement and coordination protocols.

c) Automation and Intelligence

ML-guided coverage closure could leverage execution pattern analysis to automatically generate targeted stimuli for specific verification scenarios. Intelligent coverage closure strategies might employ machine learning techniques to predict which test scenarios will most effectively improve coverage metrics.

Self-adapting verification frameworks could automatically tune parameters based on design characteristics and verification progress, reducing

manual configuration overhead while optimizing verification efficiency.

Table 4: Performance and Quality Metrics Comparison [8, 9]

Metric Category	Traditional Methods	Proposed Framework	Improvement Factor
Coverage Completeness	Directed + Random testing	SIMT-aware generation	Enhanced scenario exploration
Debug Efficiency	Manual analysis	Integrated monitoring	Accelerated issue resolution
Test bench Reusability	Project-specific design	Parameterized architecture	Cross-generation deployment
Setup Time	Custom development	Configuration-based	Reduced initial overhead
Bug Detection Timing	Post-silicon discovery	Pre-silicon identification	Earlier validation cycles

X. Conclusion

The development of a scalable UVM-based test bench architecture for GPU compute units addresses critical gaps in contemporary semiconductor verification methodologies, providing the industry with a systematic approach to validating massively parallel architectures. This comprehensive framework successfully bridges the divide between established UVM practices and the unique requirements of GPU verification, delivering measurable benefits in coverage completeness, debug efficiency, and verification reusability across diverse architectural configurations.

Through its SIMT-aware stimulus generation, integrated memory hierarchy modeling, and parameterized design approach, the framework demonstrates substantial improvements in pre-silicon validation quality while reducing overall verification development overhead. The architecture's extensibility to AI accelerators, chiplet-based designs, and future computing paradigms positions it as a valuable long-term asset for semiconductor organizations seeking to maintain verification quality as architectural complexity increases. Industry adoption of this approach promises to elevate GPU verification practices from ad-hoc, project-specific solutions toward standardized, reusable methodologies that can scale with the demanding requirements of next-generation parallel computing architectures.

References Références Referencias

1. Wilson Research Group, "2022 Functional Verification Study," Siemens EDA, 2022. Available: https://verificationacademy.com/topics/planning-measurement-and-analysis/wrg-industry-data-and-trends/2022-functional-verification-study/
2. Top 500 .org, "GPU Share of TOP 500 Super computers," November 2023. Available: https://technologymagazine.com/articles/how-nvidia-dominated-the-top500-list-with-ai-supercomputers
3. Accellera Systems Initiative, "Universal Verification Methodology (UVM) 1.2 User's Guide," October 8, 2015. Available: https://www.accellera.org/images/downloads/standards/uvm/UVM_Class_Reference_Manual_1.2.pdf
4. Bergeron, J., et al., "Verification Methodology Manual for System Verilog," Springer, 2023. Available: https://link.springer.com/book/10.1007/978-1-4419-0489-9
5. Li, A., "GPU performance modeling and optimization," PhD Thesis, Technische Universiteit Eindhoven, 2016. Available: https://research.tue.nl/files/39759895/20161018_Li.pdf
6. NVIDIA Corporation, "CUDA C++ Programming Guide," Version 12.0, 2023. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/
7. Arora, S., "Key Performance Test Metrics to Track," Perforce Blaze Meter, 2023. Available: https://www.blazemeter.com/blog/key-test-metrics-to-track
8. Oye, E., et al., "A Comparative Study on Simulation-Based Versus Emulation-Based Verification for High-Performance AI Accelerators," Research Gate, 2024. Available: https://www.researchgate.net/publication/392131135
9. IEEE Standards Association, "IEEE Standard for System, Software, and Hardware Verification and Validation," IEEE Std 1012-2016. Available: https://ieeexplore.ieee.org/document/8055462