The Anatomy of an Efficient Blackwell GEMM

Microbenchmarking and Hardware-Software Co-Design for Edge Computing

Antonio Moral Villarín

2026-01-01

Table of contents

v

Acknowledgements **vi**

Abstract **vii**

1 Chapter 1 – Introduction **1**

1.1 Motivation and Context: The Need for Hardware–Software Co-Design 1
1.2 Challenges in Efficient Compute for AI and Edge Applications 2
1.2.1 Increasing Model Complexity and Computational Growth 2
1.2.2 Memory Bandwidth and Capacity Constraints 3
1.2.3 Communication Overheads in Distributed and Heterogeneous Systems . . 3
1.2.4 Numerical Precision, Stability, and Robust Execution 3
1.2.5 Energy Efficiency and Thermal Constraints 4
1.2.6 Lack of Transparent, Fine-Grained Performance Characterization 4
1.2.7 Summary 4
1.3 Objectives and Scope of the Thesis 5
1.4 Methodology Overview 5
1.5 Structure of the Thesis 5

2 Chapter 2 – Background and Related Work **6**

2.1 Evolution of GPU Architectures: From Volta to Blackwell 6
2.2 Hardware–Software Co-Design: Principles and Applications 6
2.3 General Matrix-Matrix Multiplication (GEMM) in AI Workloads 6
2.4 Domain-Specific Languages (DSLs) for GPU Programming 6
2.5 Relevant Publications and Tools (NVIDIA Research, Citadel, JAX Scaling Book, etc.) 6

3 Chapter 3 – Architecture Comparison: Hopper vs Blackwell **7**

3.1 Overview of Hopper Architecture 7
3.2 Overview of Blackwell Architecture 7
3.3 Key Innovations in Blackwell 7
3.3.1 Ultra Tensor Cores and New Precision Formats (FP8, FP4) 7
3.3.2 Transformer Engine and FP4 Micro Scaling 7
3.3.3 Multi-Die Chip Design and Interconnect (NVLink, NVSwitch) 7
3.3.4 Memory System: HBM3e, L2 Cache, and Shared Memory 7
3.4 Performance/Watt and Area Efficiency Considerations 7
3.5 Summary of Architectural Differences 7

4 Chapter 4 – Metrics for GPU Efficiency **8**

4.1 Performance per Watt 8
4.2 Compute Throughput by Data Type 8
4.3 Memory Bandwidth and Arithmetic Intensity 8
4.4 Power, Thermal Design, and Silicon Area Constraints 8
4.5 Efficiency Bottlenecks: From Memory Bound to Compute Bound 8

5 Chapter 5 – Programming Models for Modern GPUs 9
5.1 Introduction to GPU DSLs for Performance 9
5.2 Triton 9
5.3 ThunderKittens (TK) 9
5.4 TileLang 9
5.5 Cute and CUTLASS 9
5.6 Gluon 9
5.7 Pallas and the JAX ML Scaling Framework 9
5.8 Summary: DSLs as Enablers of Architectural Efficiency 9

6 Chapter 6 – Methodology and Experimental Setup 10
6.1 Objectives of Benchmarking 10
6.2 Hardware Platforms and Specifications 10
6.2.1 Hopper H100 10
6.2.2 Blackwell B200 10
6.3 Software Tools and Libraries Used 10
6.4 Microbenchmark Design: GEMM Kernel Implementations 10
6.5 Measurement Techniques 10
6.5.1 Throughput (FLOP/s) 10
6.5.2 Power Consumption and Efficiency 10
6.5.3 Memory Bandwidth 10
6.6 Ensuring Fairness and Reproducibility 10

7 Chapter 7 – Results and Discussion 11
7.1 Performance Comparison Across Data Types 11
7.2 Analysis of Performance per Watt 11
7.3 Memory Bandwidth Observations 11
7.4 Impact of TMA (Tensor Memory Accelerator) 11
7.5 Roofline Analysis: Compute vs Memory Bound 11
7.6 Real-World Relevance: Case Study on Transformer Inference/Training 11
7.7 Discussion of Bottlenecks and Architectural Impact 11

8 Chapter 8 – Conclusions and Future Work 12
8.1 Summary of Findings 12
8.2 Implications for Hardware–Software Co-Design 12
8.3 Relevance to Edge Computing 12
8.4 Future Work and Doctoral Research Directions 12

9 References 13

10 Appendices 14
10.1 Appendix A: Experimental Scripts and Kernel Listings 14
10.2 Appendix B: Extended Benchmark Results 14
10.3 Appendix C: TMA and GEMM Intrinsics Documentation 14

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisors at the University of Sevilla...

Abstract

Here you will write the abstract of your thesis. It should summarize the motivation (hardware-software co-design), the methodology (microbenchmarking B200 vs H100), and the main findings regarding efficiency and edge computing implications.

Keywords: NVIDIA Blackwell, GEMM, Edge Computing, Hardware-Software Co-design, Tensor Cores, FP4.

1 Chapter 1 – Introduction

1.1 Motivation and Context: The Need for Hardware–Software Co-Design

The rapid growth of computational demands in artificial intelligence, scientific workloads, and large-scale data analytics has reshaped the landscape of system design. Modern applications such as trillion-parameter language models, high-fidelity simulations, and latency-constrained inference place unprecedented pressure on both compute throughput and memory bandwidth. Within this context, GPUs have evolved into the core computational substrate of contemporary high-performance and AI-centric infrastructures. The architectural transition from NVIDIA Hopper to NVIDIA Blackwell embodies not only a generational increase in performance, but also a deeper paradigm shift: system efficiency is determined by the joint design of hardware and software rather than by hardware improvements alone.

Hopper introduced features such as the Transformer Engine with FP8 support, mixed-precision Tensor Cores, and a memory hierarchy aimed at balancing HPC and AI workloads. Blackwell significantly extends this trajectory with a dual-die design of approximately 208 billion transistors, second-generation Transformer Engine capabilities, FP4 precision formats, and up to 8 TB/s of HBM3E memory bandwidth in the B200 GPU. These hardware innovations target real-time inference at unprecedented scale and emphasize energy efficiency per generated token—an increasingly important metric in large-scale AI deployments.

Such developments exemplify the principles of hardware–software co-design, where architectural decisions and software mechanisms evolve iteratively. Features like quantization formats, sparsity support, collective communication engines, or fused compute kernels demonstrate that performance emerges from coordinated advances across numerical methods, compiler frameworks, runtime systems, and workload-specific optimizations. Historically, breakthroughs such as mixed-precision training only yielded practical performance gains when software frameworks adopted new scaling rules, kernel fusion strategies, and stable optimization methods. Blackwell continues this trend: the performance benefits associated with FP4 formats or enhanced tensor engines depend directly on compiler support, quantization robustness, and the model architectures that can exploit such precision regimes.

In practice, effective performance in AI training and inference is known to depend less on theoretical peak FLOPs and more on how effectively software layers—kernels, communication libraries, frameworks, and model implementations—map onto the hardware. The literature on scalable system design, such as the Scaling Book, emphasizes that identifying whether a workload is compute-bound, memory-bound, or communication-bound is fundamental for designing efficient execution strategies. Achieving compute-bound operation requires an alignment among tensor core utilization, memory hierarchy behavior, kernel scheduling, and parallelization strategies. Microbenchmarking, therefore, becomes essential: only through systematic characterization of primitive operations can one understand how real workloads will stress the hardware.

The Blackwell architecture provides an excellent case study to illustrate these principles. The architectural differences with respect to Hopper—new precision formats, a redesigned cache and memory hierarchy, increased NVLink bandwidth, and changes in warp scheduling—directly affect kernel behavior. Initial low-level measurements, such as those documented in open-source investigations like *blackwell-anatomy*, reveal divergences in latency, achieved throughput, memory access patterns, and tensor core utilization relative to Hopper. These observations highlight that software optimized for Hopper does not trivially translate to optimal performance on Blackwell; instead, kernels require adaptation to exploit the newer architectural features effectively. This interdependence underscores the necessity of a hardware–software co-design mindset.

Although the present thesis focuses on data-center-class GPUs, the underlying principles are equally relevant to edge computing systems—an area of future research interest. Edge accelerators must operate under stringent power and latency constraints while supporting increasingly complex AI workloads. Techniques such as low-precision arithmetic, operator fusion, and communication-aware scheduling often emerge first in large GPUs and later transition to edge architectures. Understanding how Hopper and Blackwell respond to distinct microbenchmarks, and how software must adapt to each generation, provides the conceptual foundation necessary to transfer these insights to constrained environments.

In this context, the motivation for the present work is twofold. First, there is a need for a rigorous and independent microbenchmarking study of the NVIDIA B200 GPU that characterizes its architectural behavior beyond vendor-reported metrics. Vendor whitepapers offer high-level guidance but rarely expose fine-grained performance phenomena. A systematic benchmarking effort can reveal architectural bottlenecks, utilization ceilings, and workload-dependent behaviors. Second, by framing this study explicitly within the lens of hardware–software co-design, the thesis aims to articulate not only how Blackwell differs from Hopper, but why such differences matter for real workloads. The goal is to demonstrate, with empirical evidence, the importance of co-design as a methodology and to build a skillset directly applicable to future research in edge computing environments, where hardware constraints and software complexity converge most acutely.

1.2 Challenges in Efficient Compute for AI and Edge Applications

The rapid evolution of artificial intelligence, particularly in deep learning and large-scale foundation models, has revealed fundamental limitations in current compute architectures. Although GPU-accelerated systems have become the dominant platform for AI workloads, achieving high efficiency across diverse operating conditions—ranging from hyperscale training clusters to power-constrained edge devices—remains a significant challenge. Efficiency, in this context, encompasses not only peak theoretical throughput, but also sustained performance, energy efficiency, memory utilization, communication overheads, and adaptability to emerging model architectures. The following subsections outline the primary challenges that motivate this thesis and contextualize the need for a deeper architectural and microarchitectural understanding of modern GPUs such as NVIDIA Hopper and Blackwell.

1.2.1 Increasing Model Complexity and Computational Growth

The computational requirements of state-of-the-art AI models have grown at a superlinear rate. Transformer-based architectures dominate modern AI, and their scaling behavior—both in parameters and sequence length—creates workloads that stress every component of the compute stack. Larger embedding dimensions, attention mechanisms with quadratic complexity, and extended context windows impose significant pressure on compute units, memory bandwidth, and cache hierarchies.

In large data centers, the primary challenge lies in providing sufficient compute density and communication bandwidth to sustain multi-node training without falling into communication-bound regimes. At the edge, however, the challenge is even more severe: only a fraction of the necessary compute and memory capacity is available, and energy consumption must be strictly bounded. This disparity underscores the importance of understanding how architectural features, such as the tensor engines in Hopper or the FP4-capable Transformer Engine in Blackwell, behave under different numerical formats and model configurations.

1.2.2 Memory Bandwidth and Capacity Constraints

As GPUs evolve, memory bandwidth has become the dominant bottleneck for many AI workloads. Even when raw FLOPs increase significantly from one architecture to the next, memory bandwidth often scales at a slower pace, creating a widening "memory wall." Architectures such as Hopper and Blackwell attempt to mitigate this challenge through:

- Wider HBM memory stacks (HBM2e, HBM3, HBM3E),
- Larger shared memory and L2 cache capacities,
- More flexible residency and partitioning of on-chip memory,
- Advanced prefetching and compression mechanisms.

However, achieving near-peak memory throughput in practice requires software techniques such as kernel fusion, optimized tensor layouts, asynchronous memory operations, and reduced-precision formats that reduce memory footprint. In edge environments, limited DRAM capacity and lower memory bandwidth magnify these challenges, making efficient dataflow design indispensable.

1.2.3 Communication Overheads in Distributed and Heterogeneous Systems

Multi-GPU and multi-node scaling is essential for training and deploying large AI models. Even with the sophisticated interconnects available in modern systems—NVLink, NVSwitch, InfiniBand—the performance of distributed workloads is frequently dominated by communication rather than computation. Training efficiency can degrade sharply if model parallelism, pipeline parallelism, and collective operations are not carefully tailored to the hardware topology.

Furthermore, heterogeneity introduces additional complexity: edge systems often incorporate mixtures of CPUs, GPUs, NPUs, and embedded accelerators, each with distinct capabilities and communication characteristics. Efficiently mapping workloads to such heterogeneous platforms requires awareness of compute intensity, data locality, and the cost of synchronization across devices.

1.2.4 Numerical Precision, Stability, and Robust Execution

Precision scaling is one of the most powerful levers for improving efficiency. Hopper introduced FP8 formats to accelerate transformer-based training; Blackwell expands this paradigm by enabling FP4 inference at scale. While lower-precision arithmetic drastically reduces memory bandwidth requirements and increases compute density, it also introduces challenges:

- Maintaining numerical stability during training and inference,
- Adjusting quantization-aware algorithms,
- Ensuring robustness under adversarial or high-variance inputs,
- Designing kernels and runtimes capable of exploiting these formats without degradation.

As precision becomes a first-class design parameter, co-designing numerics, kernels, and architectures becomes essential.

1.2.5 Energy Efficiency and Thermal Constraints

In both hyperscale and edge settings, energy efficiency is central. For data centers, energy consumption directly determines operational cost and environmental impact. For edge devices, energy and thermal constraints determine feasibility. AI workloads, particularly those involving large matrix multiplications, are highly power demanding, and maintaining sustained performance requires careful dynamic management of:

- Voltage and frequency scaling,
- Thermal headroom,
- Active cooling capabilities,
- Load balancing across compute units.

Next-generation architectures such as Blackwell seek to increase performance-per-watt, yet achieving this in real workloads requires that software align with the hardware's optimal operating points.

1.2.6 Lack of Transparent, Fine-Grained Performance Characterization

Although GPU vendors publish extensive whitepapers, many architectural behaviors remain undocumented or only indirectly observable. Effective optimization—especially for edge devices or custom deployments—requires fine-grained insights into:

- Actual tensor core throughput across precisions,
- Memory coalescing patterns and load/store latencies,
- Scheduling policies,
- Warp-level synchronization costs,
- Interplay between kernel shapes and occupancy.

This gap motivates independent microbenchmarking, such as the approach pursued in this thesis and exemplified by investigations like *blackwell-anatomy*, which reveal empirically how architectural changes impact real workloads.

1.2.7 Summary

Efficient compute for AI and edge applications requires navigating a complex landscape of architectural constraints, communication patterns, numerical formats, and workload characteristics. The evolution from Hopper to Blackwell embodies many of these challenges and provides a concrete opportunity to study how architectural innovations interact with software design choices. By conducting a systematic microbenchmarking analysis of Blackwell and comparing it with its predecessor, this thesis aims to contribute to a deeper understanding of the hardware–software co-design principles required for achieving high efficiency in modern AI systems.

1.3 Objectives and Scope of the Thesis

1.4 Methodology Overview

1.5 Structure of the Thesis

2 Chapter 2 – Background and Related Work

2.1 Evolution of GPU Architectures: From Volta to Blackwell

2.2 Hardware–Software Co-Design: Principles and Applications

2.3 General Matrix-Matrix Multiplication (GEMM) in AI Workloads

2.4 Domain-Specific Languages (DSLs) for GPU Programming

2.5 Relevant Publications and Tools (NVIDIA Research, Citadel, JAX Scaling Book, etc.)

3 Chapter 3 – Architecture Comparison: Hopper vs Blackwell

3.1 Overview of Hopper Architecture

3.2 Overview of Blackwell Architecture

3.3 Key Innovations in Blackwell

3.3.1 Ultra Tensor Cores and New Precision Formats (FP8, FP4)

3.3.2 Transformer Engine and FP4 Micro Scaling

3.3.3 Multi-Die Chip Design and Interconnect (NVLink, NVSwitch)

3.3.4 Memory System: HBM3e, L2 Cache, and Shared Memory

3.4 Performance/Watt and Area Efficiency Considerations

3.5 Summary of Architectural Differences

4 Chapter 4 – Metrics for GPU Efficiency

4.1 Performance per Watt

4.2 Compute Throughput by Data Type

4.3 Memory Bandwidth and Arithmetic Intensity

4.4 Power, Thermal Design, and Silicon Area Constraints

4.5 Efficiency Bottlenecks: From Memory Bound to Compute Bound

5 Chapter 5 – Programming Models for Modern GPUs

5.1 Introduction to GPU DSLs for Performance

5.2 Triton

5.3 ThunderKittens (TK)

5.4 TileLang

5.5 Cute and CUTLASS

5.6 Gluon

5.7 Pallas and the JAX ML Scaling Framework

5.8 Summary: DSLs as Enablers of Architectural Efficiency

6 Chapter 6 – Methodology and Experimental Setup

6.1 Objectives of Benchmarking

6.2 Hardware Platforms and Specifications

6.2.1 Hopper H100

6.2.2 Blackwell B200

6.3 Software Tools and Libraries Used

6.4 Microbenchmark Design: GEMM Kernel Implementations

6.5 Measurement Techniques

6.5.1 Throughput (FLOP/s)

6.5.2 Power Consumption and Efficiency

6.5.3 Memory Bandwidth

6.6 Ensuring Fairness and Reproducibility

7 Chapter 7 – Results and Discussion

7.1 Performance Comparison Across Data Types

7.2 Analysis of Performance per Watt

7.3 Memory Bandwidth Observations

7.4 Impact of TMA (Tensor Memory Accelerator)

7.5 Roofline Analysis: Compute vs Memory Bound

7.6 Real-World Relevance: Case Study on Transformer Inference/Training

7.7 Discussion of Bottlenecks and Architectural Impact

8 Chapter 8 – Conclusions and Future Work

8.1 Summary of Findings

8.2 Implications for Hardware–Software Co-Design

8.3 Relevance to Edge Computing

8.4 Future Work and Doctoral Research Directions

9 References

10 Appendices

10.1 Appendix A: Experimental Scripts and Kernel Listings

10.2 Appendix B: Extended Benchmark Results

10.3 Appendix C: TMA and GEMM Intrinsics Documentation