ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

"Evaluation of gem5-based Modeling Frameworks for Heterogeneous SoC Designs with Domain-Specific Accelerators."

Αργυρώ Α. Ριτσογιάννη

Επιβλέπων ***(ή Επιβλέπουσα ή Επιβλέποντες)*****:** **Δημήτρης Γκιζόπουλος,** Καθηγητής

ΑΘΗΝΑ

09/24

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Evaluation of gem5-based Simulation Frameworks for Heterogeneous System-on-Chip Designs with Domain-Specific Accelerators

Αξιολόγηση Εργαλείων Προσομοίωσης Ετερογενών Σχεδιάσεων Συστημάτων-σε-Τσιπ με Επιταχυντές Συγκεκριμένου Τομέα

Αργυρώ Α. Ριτσογιάννη

Α.Μ.: CS3190007

ΕΠΙΒΛΕΠΩΝ (ή ΕΠΙΒΛΕΠΟΝΤΕΣ): **Δημήτρης Γκιζόπουλος,** Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ (εάν υπάρχει)**:**
Γκιζόπουλος Δημήτρης, Καθηγητής
Πασχάλης Αντώνης, Καθηγητής
Παπαδημητρίου Γεώργιος, Διδάκτωρ Πληροφορικής

09/24

ΠΕΡΙΛΗΨΗ

Στον τομέα της αρχιτεκτονικής υπολογιστών, η χρήση επιταχυντών υλικού σε σύγχρονα συστήματα καθίσταται κρίσιμη για την επίτευξη της καλύτερης απόδοσης. Αυτή η διπλωματική εξετάζει διάφορα εργαλεία προσομοίωσης που βασίζονται στον μικροαρχιτεκτονικό προσομοιωτή gem5 ή μπορούν να ενταχθούν σε αυτόν, σχεδιασμένα να δοκιμάζουν διαφορετικές σχεδιάσεις System-on-Chip (SoC) που περιλαμβάνουν συγκεκριμένους επιταχυντές υλικού για συγκεκριμένες εργασίες. Ελέγχει πόσο καλά αυτά τα πλαίσια μπορούν να μιμηθούν ετερογενή SoC του πραγματικού κόσμου, εστιάζοντας σε παράγοντες όπως το πόσο γρήγορα εκτελείται η προσομοίωση, πόσο προσαρμόσιμα είναι και πόσο εύκολα είναι στη χρήση τους. Η πτυχιακή αυτή διερευνά επίσης πώς διαφορετικές ρυθμίσεις και μέθοδοι μοντελοποίησης στον gem5 επηρεάζουν την ομαλή ενσωμάτωση αυτών των συγκεκριμένων επιταχυντών στα συστήματα. Τα ευρήματα παρέχουν χρήσιμες πληροφορίες για τη λήψη ενημερωμένων αποφάσεων σχετικά με τα πλαίσια προσομοίωσης που θα χρησιμοποιηθούν, βοηθώντας στην αποτελεσματική αξιολόγηση και ανάπτυξη προηγμένων σχεδίων SoC.

ABSTRACT

In the field of computer architecture, using hardware accelerators in modern systems is becoming crucial for achieving the best performance. This thesis surveys various simulation frameworks that are based on or can be integrated into the microarchitectural simulator gem5, designed to test different System-on-Chip (SoC) designs that include specific hardware accelerators for certain tasks. It analyzes how well these frameworks can mimic real-world heterogeneous SoCs, focusing on aspects such as the simulation speed, ease of customization, ease of use. The study also explores how different settings and modeling methods in gem5 affect the smooth addition of these specific accelerators into systems. The findings provide useful information for making informed decisions about which simulation frameworks to use, helping in the effective evaluation and development of advanced SoC designs.

SUBJECT AREA: Computer Architecture

KEYWORDS: Hardware Accelerators, gem5 Simulation Framework, Heterogeneous System-on-Chip, Domain-Specific Accelerators, Computer Architecture, Simulation Framework Comparison, gem5-aladdin, gem5-SALAM, Maestro, SMAUG, gem5-x

THANKS

I would like to thank my prof. Dimitris Gizopoulos for suggesting the idea of my thesis and for giving me the opportunity to work on such an interesting topic.

I would also like to thank my supervisor, PostDoc Researcher George Papadimitriou, for the continuous support and the help towards completing this thesis by providing help and suggestions throughout it.

October 2024

CONTENTS

Πίνακας περιεχομένων

List of Figures

List of Tables

PROLOGUE

This thesis examines gem5-based and gem5-compatible simulation frameworks in the context of modern computing. It aims to evaluate these tools in relation to heterogeneous SoC designs, offering a comprehensive understanding of their role in contemporary computer architecture. The work is rooted in both technical analysis and academic inquiry, reflecting the broader pursuit of knowledge within the academic community.

1. INTRODUCTION

1.1 Overview of the importance of accelerators in modern computing

In the dynamic landscape of modern computing, the rapid evolution of System-on-Chip (SoC) designs underscores the indispensable role of specialized hardware accelerators, which, meticulously crafted for specific computational tasks, stand as pivotal components driving the performance and functionality of modern architectures [13]. Unlike general-purpose processors, accelerators are tailored to execute specific workloads efficiently, ranging from complex scientific simulations to data-intensive algorithms [33].

As applications and workloads diversify, the demand for optimized computational solutions intensifies. Herein lies the significance of accelerators within the intricate field of modern computing. They serve as dedicated engines, fine-tuned to excel in particular tasks, thereby unlocking unparalleled performance gains and energy efficiency. In the quest for enhanced computational capabilities, SoC designs increasingly integrate domain-specific accelerators (DSA), ushering in an era where customization and specialization become paramount.

However, the integration of accelerators introduces a complex interplay of architectural elements, necessitating sophisticated simulation frameworks for thorough evaluation and design space exploration. This study deepens within the critical task of assessing the variety of gem5-based simulation frameworks, which are inherently designed to aid in understanding the functionality and performance of heterogeneous architectures incorporating domain-specific accelerators. Gem5 [2], a well-established open-source microarchitectural simulator, emerges as a linchpin, providing a flexible and extensible platform for computer architects and developers to explore the intricate nuances of these specialized hardware components.

By seamlessly integrating DSAs into heterogeneous SoC designs, designers can tailor their architectures to optimize performance for specific workloads. This customization not only enhances computational efficiency but also contributes to a reduction in power (and energy) consumption and overall area utilization [34]. The intersection of accelerators and SoC designs encapsulates a transformative paradigm in computing, where adaptability and specialization converge to meet the evolving demands of diverse applications.

In essence, this study embarks on an exploration of the symbiotic relationship between accelerators and SoC designs. It scrutinizes the intricate balance between performance gains and architectural complexities, shedding light on the imperative need for efficient simulation frameworks. Through this perspective, the thesis highlights the importance of accelerators in modern computing, showing how they drive innovation in the field.

1.2 The significance of having a framework for designing and evaluating accelerators

In heterogeneous computing, dedicated frameworks for modeling and validating accelerator designs are paramount, particularly as System-on-Chip designs rapidly evolve. These frameworks are crucial for several reasons [13][31]:

- **Complexity Management**: Modern SoCs incorporate a variety of components, including CPUs, GPUs, and specialized accelerators, each with unique performance and energy characteristics. A structured framework aids in managing this complexity, allowing designers to simulate and analyze the behavior of these components in a unified environment.

- **Domain-specific Acceleration**: As computing demands diversify across applications like machine learning, image processing, and scientific simulations, the need for accelerators optimized for specific tasks has grown. Dedicated simulation frameworks enable the exploration of novel accelerator designs before committing to expensive fabrication processes.

- **Integration Challenges**: Integrating accelerators into heterogeneous architectures presents challenges, including memory access patterns, data movement, interconnects and communication with general-purpose processors. Frameworks that model these aspects can help identify bottlenecks and optimize system architecture for balanced performance and energy efficiency.

- **Performance Validation**: Through simulation, designers can validate the expected performance improvements offered by accelerators within the context of a full system. This is critical for ensuring that the inclusion of an accelerator leads to tangible benefits under realistic workloads.

- **Rapid Iteration for design space exploration**: Dedicated frameworks support rapid prototyping and iteration of accelerator designs, enabling a more agile development process. Designers can quickly assess the impact of changes to architecture, algorithms, or system integration strategies.

At the forefront of these simulation frameworks stands gem5, an open-source CPU simulator widely adopted for its flexibility and extensibility [2]. This transformative capability aligns with the ever-growing demand for specialized solutions in the face of diverse computational workloads. The primary function of a simulation framework is to create a virtual environment that emulates the intricacies of real-world scenarios. The framework serves as a sandbox, facilitating the exploration of diverse architectural parameters and configurations.

A major benefit of using a simulation framework is the broad early design space exploration using thorough evaluation of various metrics, including simulation output results, efficiency, configurability, and ease of use. These metrics contribute to a nuanced understanding of how well a framework can accurately model and emulate heterogeneous SoC designs

featuring domain-specific accelerators. Simulation frameworks play a crucial role in evaluating the impact of configuration choices and modeling techniques within gem5 on overall simulation performance. This investigation becomes vital in the ongoing efforts to design efficient simulation methodologies for next-generation SoC architectures. Understanding the balance between configurability and performance is paramount for achieving optimal results in accelerator design and integration.

The challenges associated with incorporating different domain-specific accelerators into simulation frameworks are also explored in this context. Addressing issues such as integration SoCs and optimization performance is crucial for ensuring that the simulated environment accurately reflects the expected behavior of accelerators in real-world scenarios. These challenges underscore the nuanced nature of accelerator integration and the importance of a robust simulation framework in overcoming hurdles in the design and evaluating phase.

In the culmination of this research, specific gem5-based frameworks, namely gem5-Aladdin and gem5-SALAM, take center stage in the demonstration of metric outcomes. Distinguished by their user-friendly nature, these frameworks showcase how the choice of a simulation tool can impact the ease with which designers and researchers navigate the complexities of heterogeneous SoC designs.

In conclusion, the significance of having a framework for designing and evaluating accelerators extends beyond mere technical functionality. It encapsulates a paradigm shift in the approach to SoC evolution, providing a structured pathway for developers to innovate, optimize, and adapt to the ever-changing demands of modern computing. A robust simulation framework is not just a tool it is a catalyst for progress, empowering designers to chart the course of next-generation SoC architectures with precision and efficiency.

2. BACKROUND AND SIMULATION FRAMEWORKS ANALYSIS

In the domain of heterogeneous SoC, five frameworks, namely, gem5-Aladdin [32], SMAUG [35], gem5-SALAM [28], gem5-X[24,25], and MAESTRO[17] have emerged as prominent gem5-based simulation frameworks that provide valuable tools for modeling and evaluating domain-specific accelerators integrated to a host CPU. These frameworks offer a range of features and capabilities to support the design and optimization of accelerators. The thesis examines their architectural aspects, similarities, differences, advantages, disadvantages, and the functionalities they support.

2.1 gem5-Aladdin

gem5-Aladdin stands out for its integration with the Aladdin framework [37], enabling cycle-level simulation and in-depth analysis of accelerators [32]. It offers a comprehensive modeling infrastructure and supports the evaluation of accelerator performance, power consumption, and area characteristics. gem5-Aladdin creates a unified environment for accelerator simulation within a gem5-based SoC architecture. The gem5 simulator provides the system-level context, modeling the various components of the SoC such as Processing Units like ARM, x86, MIPS, RISC-V, and SPARC, while the Aladdin framework specializes in modeling and analyzing domain-specific accelerators including those for machine learning, graphics, and custom digital signal processing tasks [2]. It allows for cycle-level simulation of these accelerators, providing insights into their performance, power consumption, and area (PPA) metrics within the context of the larger system simulated by gem5. Through synchronization mechanisms, gem5-Aladdin enables seamless communication and information exchange between the gem5 simulator and the Aladdin framework, which allows for capturing fine-grained details of accelerator behavior.

Data movement that is not effectively managed or accounted for in the system can consume a substantial part of the overall runtime, making the use of highly parallel datapaths redundant. Understanding the impact of emulation effects on accelerator behavior necessitates simulation infrastructures capable of modeling these heterogeneous systems. gem5-Aladdin addresses this need by modeling interactions between accelerators and CPUs, Direct Memory Access (DMA), hardware-managed caches, and virtual memory. Each of these features has implications for the behavior and performance of the accelerators. For example, Figure 1 illustrates an example of a System on Chip (SoC), featuring general-purpose cores, memory controllers, a DMA engine, and various types of fixed-function accelerators, all interconnected via the system bus.

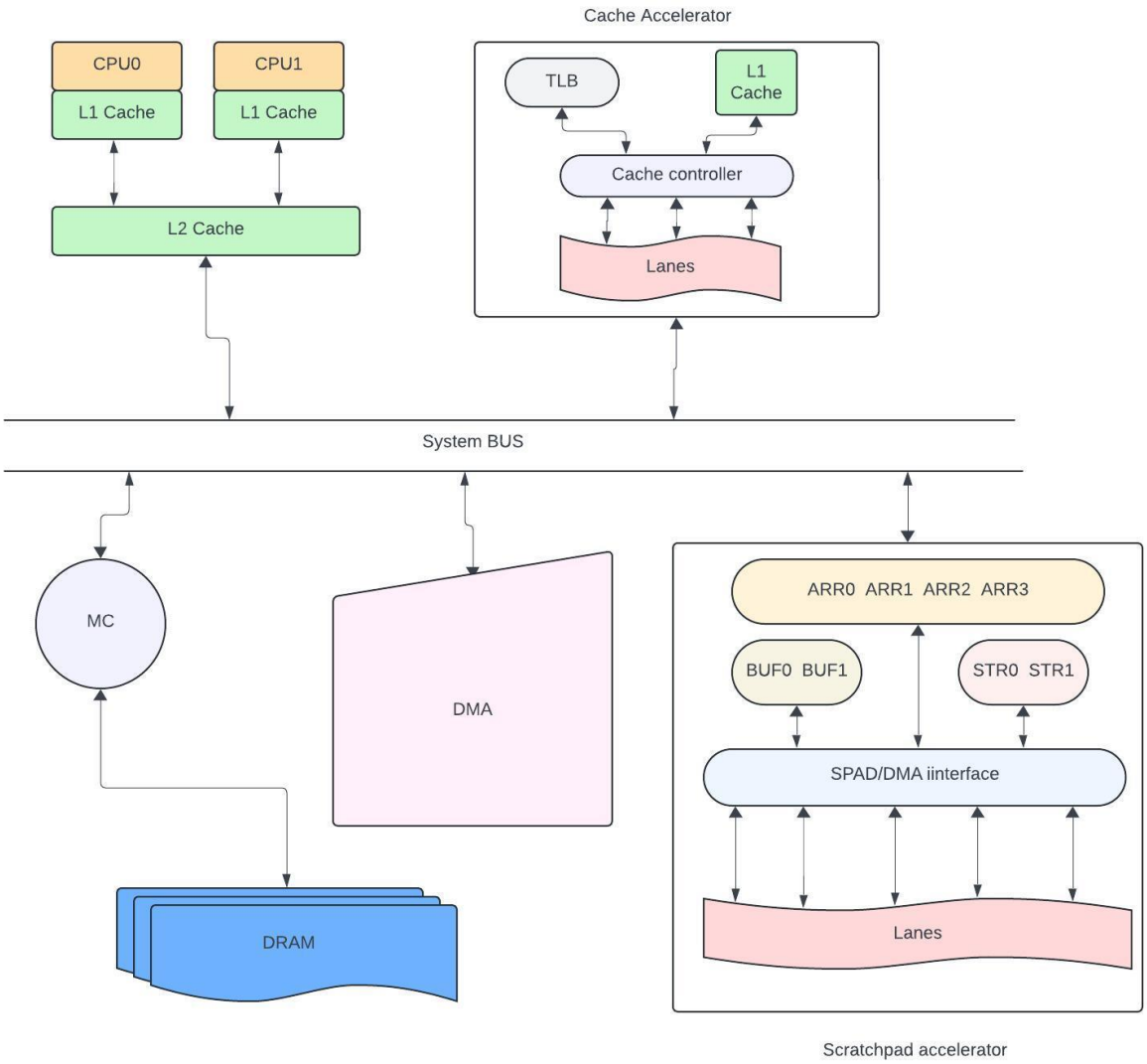


Figure 1: Example of a System on Chip (SoC)

The gem5-Aladdin framework allows accelerators to utilize the DMA engine in gem5 by inserting special function calls for data loading and storing. These calls are recognized as DMA operations by Aladdin, which then communicates with the gem5 DMA engine. An analytical model is integrated into the DMA engine to handle cache flush and invalidation latency. gem5-Aladdin effectively models a wide range of accelerator workloads, including system-level considerations, without modifying gem5's cache models. The paper discusses the design considerations when choosing between a DMA- or cache-based memory system for an accelerator, evaluating the performance of both approaches. It emphasizes the

importance of considering system-level effects, such as data movement, in accelerator design to achieve optimal performance.

The Aladdin accelerator simulator plays a crucial role in this research by modeling the power, performance, and cycle-level activity of standalone, fixed-function accelerators without the need to generate register transfer level (RTL) designs. It operates as a trace-based accelerator simulator, profiling the dynamic execution of a program and constructing a dynamic data dependence graph (DDDG) as a dataflow representation of the accelerator. The DDDG comprises LLVM IR instructions as vertices, with edges denoting true dependencies between operations. Aladdin employs common accelerator design optimizations and schedules the graph for execution through a breadth-first traversal, while considering user-defined hardware constraints. However, it is important to note that Aladdin solely focuses on the standalone datapath and local memories of accelerators. It assumes that all data has been preloaded into the local scratchpads, omitting the modeling of any interactions between accelerators and the larger system to which they belong. Scratchpad memory is a type of on-chip memory that does not employ automatic data caching mechanisms. Instead, data management (loading and eviction) is explicitly controlled by the software or hardware designer. This allows for more predictable memory access times since the latency is consistent and does not depend on the cache state or replacement policies. Scratchpads are typically found in systems where timing predictability and efficiency are paramount, such as in real-time systems or in the domain-specific accelerators that Aladdin models. Preloading data into local scratchpads is a critical strategy in the design of hardware accelerators, like those modeled by Aladdin, due to several compelling benefits it offers in terms of performance, energy efficiency, and predictability. Scratchpad memory (SPM) is a form of statically managed memory that differs from the traditional cache in several keyways, making it especially suited for the deterministic workloads often encountered in specialized computing tasks [13]. Figure 2 shows a shared scratchpad memory. The benefits of preloading data into local Scratchpads are performance prediction, energy efficiency, reduced latency, the avoidance of Cache Coherence Overheads and the Customization to Workload [31]. Please note that the proportions of the shape are not accurate and the figure serves only as an indicative representation.

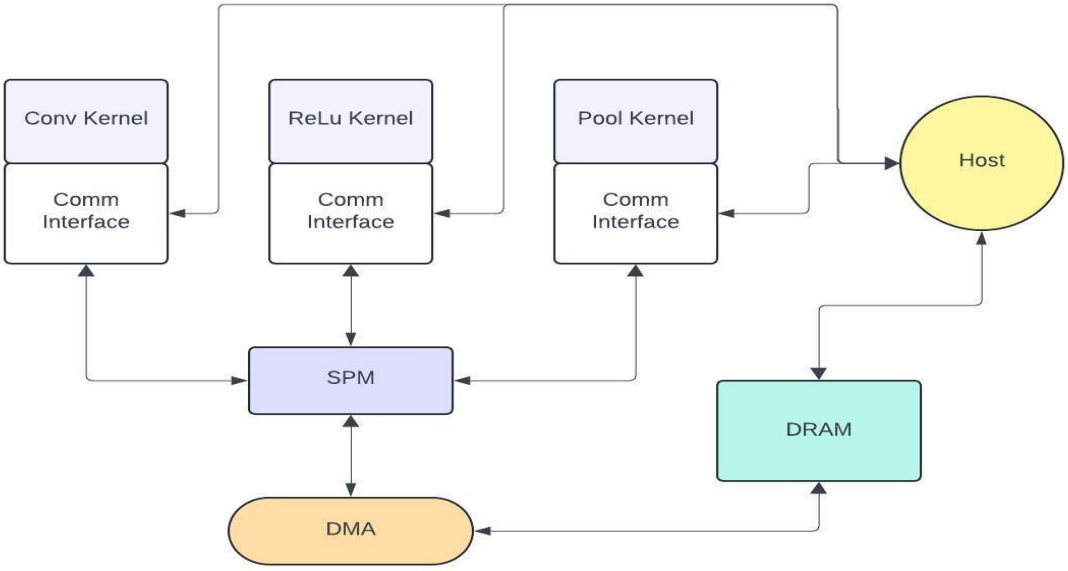


Figure *2:* Shared Scratchpad Memory

While gem5 offers support for a wide range of architectures, gem5-aladdin provides support exclusively for the x86 architecture. All the experiments that were conducted using gem5-Aladdin were in syscall emulation mode, which adequately captures the effects of system-level considerations on performance and power usage and offers the advantage of faster simulation compared to full-system simulation. Full-system simulation, which models operating system effects, was not utilized due to its scope limitations. However, certain interactions with the operating system, such as device driver to hardware interactions, are supported by the simulator through real hardware measurements and analytical inclusion in the models.

The design of gem5-Aladdin is centered around the concept of bridging the gap between hardware and software in heterogeneous systems. This integration is crucial as specialized hardware accelerators are increasingly being used alongside traditional CPU cores. gem5 offers a detailed simulation of processor cores, memory systems, and system-level architecture, providing a realistic environment for architectural exploration. Aladdin complements this by offering precise modeling of accelerators, focusing on their power consumption and performance metrics.

Unique Features and Capabilities:

- **Co-Simulation Environment**: gem5-Aladdin co-simulates both CPU and hardware accelerators, supporting x86 and syscall emulation. This is pivotal in analyzing system interactions, particularly for compute-intensive functions.
- **Flexibility and Customization**: Based on the gem5 framework, which supports various ISAs including Alpha, ARM, SPARC, MIPS, POWER, RISC-V and x86, it allows researchers to integrate custom accelerator models for specific research needs.

- **In-Depth Power and Performance Analysis**: Aladdin's integration within gem5 provides detailed insights into both power usage and performance aspects of SoC designs, critical for optimizing systems for energy efficiency and computational effectiveness.
- **User Accessibility**: Despite its sophistication, gem5-Aladdin is designed with a user-friendly interface, making it accessible to a wide range of users, from academic researchers to industrial practitioners.

Diverse Application Scenarios:

- In the domain of Artificial Intelligence, gem5-Aladdin plays a crucial role. This tool enables the simulation and evaluation of different hardware accelerators, which are essential for running AI algorithms. It assists in optimizing these systems for enhanced speed and power usage efficiency.
- Exploring Heterogeneous Computing: The framework is ideal for researching heterogeneous computing systems that combine CPUs with specialized accelerators. Researchers can experiment with different configurations to achieve optimal performance and energy consumption ratios.
- Big Data Processing: For big data applications, gem5-Aladdin can simulate how accelerators interact with memory systems to optimize data movement between global memory and local memory. This provides critical insights into designing accelerators with specialized interfaces for efficient bulk data transfers, especially in workloads like memcached[40] and database partitioning[41], where large datasets need to be processed efficiently.

The significance of gem5-Aladdin in both academic and industrial contexts is underscored by numerous high-impact publications and citations. For example, Xi et al. in [35] discuss the broader implications of accelerator-focused simulations in emerging computing paradigms. Also, Binkert et al. in [2] emphasize its role in the broader context of computer architecture research. These works illustrate the widespread adoption and importance of gem5-Aladdin in advancing the field of SoC design and architectural exploration. The framework's ability to simulate complex interactions between different system components and its detailed power-performance analysis capabilities make it an indispensable tool in the field.

2.2 SMAUG

SMAUG builds on gem5-Aladdin, which employs a preRTL approach to model the power, performance, and area of accelerator designs. Studying end-to-end behavior in simulation is crucial for designing DNN-centric SoCs, yet no DNN framework supports fast, early-stage design exploration. SMAUG addresses this gap as the first architecture-simulation friendly deep learning framework that runs in a user-level simulator. Its compatibility with gem5-

Aladdin allows for flexible SoC and memory topologies without requiring RTL for design space exploration, simplifying the research and development process.

In this paper, the authors introduce SMAUG, a gem5-based framework and the first deep neural network (DNN) framework specifically designed for the simulation of end-to-end deep learning applications. SMAUG offers a range of capabilities for evaluating DNN workloads, including diverse network topologies, easy accelerator modeling, and seamless system-on-chip integration. The paper emphasizes the importance of evaluating the full-stack performance of hardware-accelerated computing tasks like neural network inference, highlighting the need for suitable research infrastructure to address components such as data transformation and movement costs and software framework overheads.

To address the above challenges, the authors developed SMAUG, a DNN framework that can be simulated within a cycle-level SoC simulator. They demonstrate how SMAUG can optimize end-to-end performance for a wide range of DNNs by leveraging various strategies, such as optimizing the SoC-to-accelerator interfaces, exploiting multi-accelerator systems, and optimizing the software stack.

SMAUG's "end-to-end" definition encompasses the entire process, from the computation of the result to the CPU receiving the inference request. The authors break down the overall time spent on accelerator computing, data transfer to/from scratchpads, and CPU time spent in the software stack. They emphasize the importance of studying end-to-end behavior in simulation to holistically design DNN-centric SoCs, especially during the early stages of hardware design.

Figure 3 depicts the execution flow and the architecture of SMAUG. This framework consists of three major components: a Python frontend for network configuration, a C++ runtime for execution management, and a backend comprising a set of hardware-accelerated kernels. The accelerated kernels can be modeled using either Aladdin or native gem5 simulation objects, offering flexibility and control to the user. SMAUG is compatible with LLVM-based toolchains required by the Aladdin accelerator simulator and exposes gem5 APIs, with improvements made to support C++ binaries, multi-threaded workloads, and sampling.

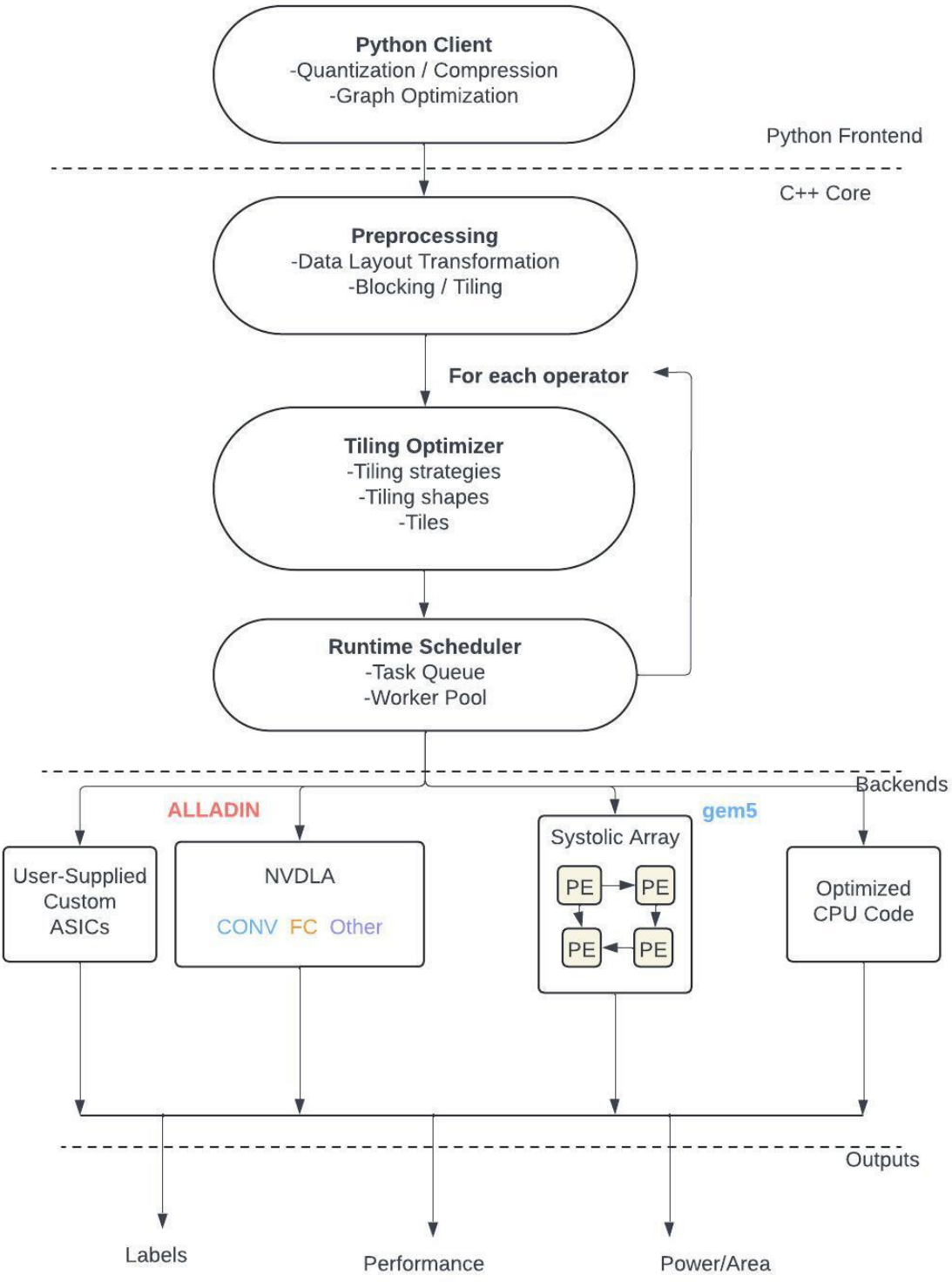


Figure 3: Smaug's execution flow

Efficient tiling schedules, maximizing data reuse, and minimizing data movement, are critical for achieving high-performance DNN computations. SMAUG supports different tiling strategies, where changing the tiling shape may impact overall runtime, particularly for operations relying on data reuse. Accelerators can be modeled in SMAUG using Aladdin, which supports various complex hardware constructs. However, from SMAUG's perspective, all accelerators export a common interface defined by dataflow, minimum tile shape, and maximum tile size.

To distribute work across multiple accelerators, SMAUG implements an accelerator worker pool and a command queue per accelerator. Tasks are assigned to the next available accelerator in the pool, while unsupported operators are executed on the CPU. SMAUG also supports dividing CPU work across multiple threads, utilizing a thread pool with round-robin scheduling. The gem5-Aladdin API exposes accelerators as threads, simplifying concurrent execution management.

SMAUG provides features such as a Python API for network configuration, support for common operators and network topologies, plug-and-play hardware accelerator implementations, and a complete software stack managing operator tiling, multi-accelerator and multi-thread scheduling, synchronization, and more. The framework enables researchers to evaluate different accelerator and SoC designs, facilitating hardware-software co-design and end-to-end system studies.

The paper presents several case studies showcasing SMAUG's capabilities and insights gained from end-to-end DNN studies. These case studies demonstrate methods to improve overall performance in various DNNs, including:

- **SoC-Accelerator Interfaces**: Enhancing CPU-accelerator coupling to achieve significant speedup and energy savings.
- **Tile-Level Parallelism**: Utilizing multiple independent accelerators to boost compute and data-transfer throughput.
- **Multithreading for Data Preparation**: Implementing multithreading in the software stack to optimize data preparation time, resulting in improved overall performance.
- **Optimizing Data Transfers**: Using a coherent interface between the accelerator and CPU to enhance data transfer speeds.
- **Software Tiling Transformations**: Reducing CPU processing time through optimized tiling techniques to improve inference latency.

Additionally, the integration of SMAUG with a state-of-the-art camera pipeline in Halide demonstrates its ability to model complex applications and identify opportunities for more efficient system design.

SMAUG offers an easy pathway for implementing new hardware accelerator models and studying end-to-end system interactions. It simplifies the development of new hardware models and enables researchers to focus on their specific interests. The framework supports user-level simulators, such as gem5 syscall-emulation mode, reducing simulation time with

sampling and handling incomplete system call emulation. SMAUG also supports multi-threading without a thread scheduler, implementing a custom thread pool.

At its core, SMAUG is a versatile and efficient platform for simulating and evaluating SoCs. It supports exploring complex architectures with CPUs, GPUs, and specialized accelerators, enabling in-depth analysis of system performance and interactions between components.

Unique Aspects of SMAUG:

- Heterogeneous System Modeling: One of the standout features of SMAUG is its ability to model and simulate heterogeneous computing systems. This includes a detailed representation of different processing elements, which is vital in an era where SoCs increasingly rely on a mix of general-purpose processors and specialized accelerators for tasks like machine learning and data processing.
- Extensibility and Flexibility: SMAUG is designed with extensibility in mind, allowing researchers to add custom models or modify existing ones. This level of customization is crucial for simulating emerging technologies and architectural innovations in SoC design.
- Performance and Power Efficiency Analysis: SMAUG provides detailed insights into both the performance and power efficiency of SoC designs. This dual focus is essential in optimizing SoCs for a balance between computational speed and energy consumption, particularly in applications like mobile devices and high-performance computing.
- Integration with Real-World Workloads: SMAUG can integrate and run real-world workloads, which allows for more realistic and practical simulation results. This feature is particularly useful for evaluating the real-world performance implications of different SoC designs.

SMAUG has been employed in various scenarios, reflecting its versatility and effectiveness in SoC simulation:

- Machine Learning Systems: SMAUG has been instrumental in simulating and optimizing SoC designs that include machine learning accelerators. This is particularly important for understanding the performance characteristics of these systems under different workload conditions.
- Energy-Efficient Computing: For applications where energy efficiency is paramount, SMAUG provides valuable insights into how different SoC components can be optimized for lower power consumption without compromising performance.
- High-Performance Computing: SMAUG is designed to handle and optimize complex SoCs in scenarios that demand high computational throughput. It ensures efficient modeling in high-performance environments, enabling the development of more powerful and efficient computing systems.

2.3 gem5-SALAM

gem5-SALAM, which stands for 'System Architecture for LLVM-based Accelerator Modeling,' is a system architecture that moves beyond traditional trace-based simulation. While not an event-driven simulator, it significantly differs from conventional trace-based approaches due to its architecture and its ability to model multiple accelerators within clusters. Trace-based simulators often lack the necessary mechanisms for efficient simulation, especially in complex multi-accelerator environments. In contrast, gem5-SALAM provides a non-trace-based simulation environment, designed to support scalable modeling of custom hardware accelerators integrated into the gem5 framework.
One of the key strengths of gem5-SALAM is its ability to enable inter-accelerator pipelining, which results in substantial improvements in end-to-end execution compared to baseline models. This feature, along with its capability to simulate various scenarios of multi-accelerator integration, makes gem5-SALAM uniquely suited to accurately model complex accelerator interactions. By overcoming the limitations of trace-based methods, gem5-SALAM offers a more dynamic and efficient simulation environment, particularly in advanced system architectures.

The system architecture has been designed with long-term scalability in mind and is fully compatible with the gem5 system framework. It offers a modular communication interface and a flexible memory hierarchy, seamlessly integrated into the gem5 ecosystem. This integration significantly simplifies the design and modeling of accelerators for emerging applications.

According to the paper, gem5-SALAM is considered one of the most efficient simulators, as evidenced by the comparative tables below. However, while it performs exceptionally well in most scenarios, there may be cases where a different simulation approach could better meet specific needs or requirements, depending on the unique demands of the system being modeled. Despite these exceptions, the paper highlights gem5-SALAM’s strengths, including its capabilities for cycle-level profiling and comprehensive full-system design space exploration in accelerator-rich environments.

gem5-SALAM's LLVM-based simulation platform allows for scalable simulation of accelerator-rich systems on SoCs. Unlike the existing platforms mentioned in this thesis, gem5-SALAM introduces a new Sim-Object as the run-time engine within the gem5 ecosystem, enabling flexible full-system simulation with multiple hardware accelerators. Users can leverage unmodified LLVM code generated from any language along with their desired accelerators and system configurations to automatically create a full system simulation. Figure 4 includes the Accelerator Model Generation process, showcasing its role within the overall system architecture. Furthermore, gem5-SALAM demonstrates significant advantages in supporting full system simulations and design space exploration for both single and multiple accelerators with varying design configurations.

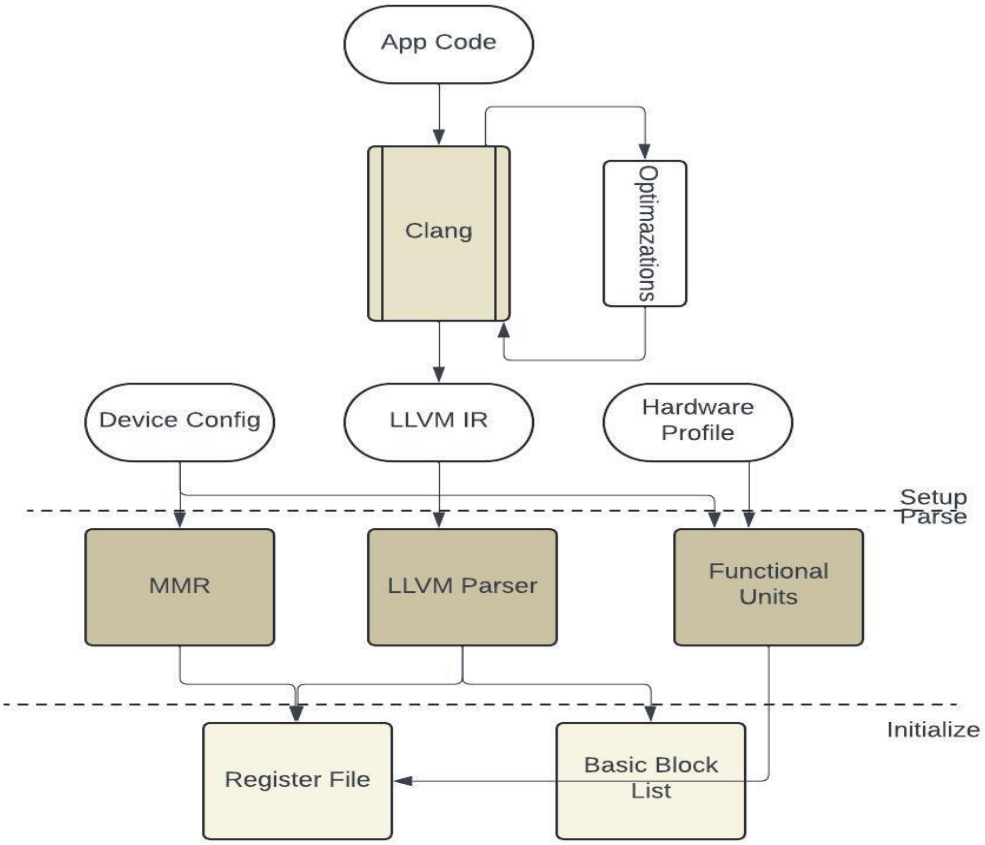


Figure 4: Accelerator model creation

gem5 currently supports various system element models, such as memory, CPUs, GPUs, and buses. However, it lacks models for application-specific hardware accelerators. To address this limitation, researchers have integrated application-specific accelerator modeling into gem5 through gem5-SALAM. This system architecture introduces accurate modeling of the datapath structure, area, and static leakage power based on algorithm-intrinsic characteristics exposed by LLVM. Additionally, gem5-SALAM incorporates cycle-accurate modeling of dynamic power consumption and timing through a dynamic LLVM-based runtime execution engine, enabling precise simulation of runtime-dependent accelerators and their interactions with other system elements. The gem5-SALAM also separates the datapath and memory infrastructure, allowing independent tuning and design space exploration, and provides a flexible system integration that directly exposes accelerator models to other system elements, fostering complex inter-accelerator communication and synchronization using pre-existing gem5 simulation constructs. The general-purpose C++/Python API for accelerator modeling in gem5-SALAM decouples computation from system communication, enabling customization and specialization to cater to various user modeling needs.

The architecture of gem5-SALAM is notable for its hybrid approach, connecting traditional CPU simulation with advanced accelerator modeling. Its core components include:

- Core gem5 Simulator: Forms the primary platform for simulating the CPU aspects of SoCs.
- Co-Simulation Capabilities: Enables simultaneous simulation of CPU components and any accelerators, essential for understanding interactions and the collective impact of SoC components.
- Modularity and Extensibility: Emphasizes modularity and extensibility, allowing for the addition of new models and adaptations to existing ones, accommodating diverse SoC designs.

gem5-SALAM brings significant enhancements to SoC simulation:

- Enhanced Precision in Accelerator Simulation: Improves accuracy and detail in simulating accelerators.
- Comprehensive System Analysis: Provides insights into interactions among SoC components and their influence on performance.
- Energy and Performance Optimization: Supports analysis and optimization of energy efficiency and computational performance.
- Facilitating Research and Innovation: Serves as a tool for researchers and developers, fostering innovation in SoC designs.
- Real-World Scenario Simulation: Simulates real-world scenarios and workloads to evaluate SoC performance under realistic conditions.

gem5-SALAM represents a significant advancement in SoC simulation, combining gem5's robust capabilities with innovative features. This integration enhances simulation accuracy

and efficiency, particularly for complex computational applications, meeting the evolving needs of modern SoC designs.

2.4 gem5-X

Gem5-X is a system-level simulation framework built upon gem5, designed to optimize many-core systems in terms of performance and power efficiency. This framework facilitates the identification of bottlenecks and the evaluation of architectural extensions, such as in-cache computing and 3D stacked High Bandwidth Memory (HBM), which are crucial for enhancing system-level architectural innovations [2]. Additionally, gem5-X has made strides in enhancing simulation speed and efficiency. By optimizing simulation algorithms and implementing more efficient data structures, it has significantly reduced the time required to model complex systems, a vital feature for conducting large-scale experiments and simulations.

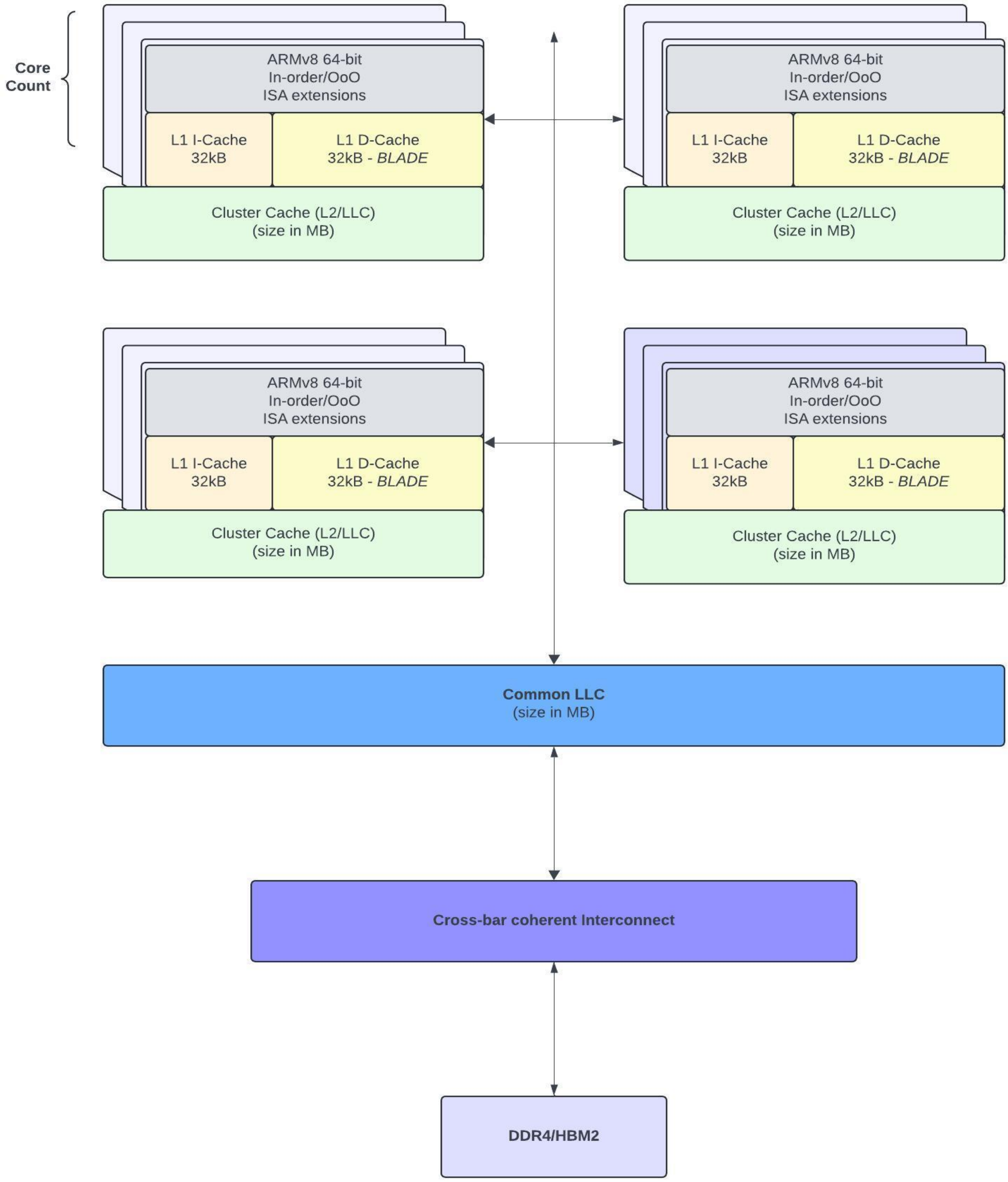


Figure 5: Gem5-X's platform

The gem5-X platform, as shown in Figure 5, is also designed for architectural exploration and optimization of real-time video analytics applications. It simulates an ARMv8 64-bit ISA

with Linux and includes application profiling via gperf to identify bottlenecks. The platform supports 9P over virtual IO for efficient file sharing and allows for the simulation of heterogeneous architectures with both in-order and out-of-order cores. It supports core clustering, custom accelerators, ISA extensions, and large-scale many-core simulations, accommodating up to 256 cores. These features make it suitable for high-performance and energy-efficient processing tasks.

To achieve efficient energy utilization and superior performance, there's a need for a system-level simulator capable of simultaneously executing multi-threaded applications on many-core systems. gem5-X, described as a "gem5-based full-system simulator with architectural eXtensions," offers a versatile platform for fast profiling, architectural exploration, and the characterization of performance-power aspects of novel architectural concepts. It seamlessly integrates applications running on modern Linux operating systems.

With the increasing importance of efficient on-chip communication in multicore processors and SoCs, gem5-X's improved models for simulating Convolutional Neural Networks (CNNs) are particularly noteworthy, offering deeper insights into the performance and scalability of these systems.

The integration capabilities of gem5-X with other simulation and modeling frameworks are where it truly stands out. For instance, its ability to integrate with power and energy modeling tools like McPAT [38] enables the simultaneous analysis of both performance and power consumption (thus also energy), a critical aspect in the design of energy-efficient computing systems. Moreover, gem5-X supports heterogeneous architectures, such as in-order and out-of-order (OoO) cores, as long with custom accelerators like an in-cache computing engine [42]. Additionally, it supports heterogeneous memory types, including DDR4 and 3D stacked HBM2, enabling the simulation of highly heterogeneous systems with a full Linux stack. To the best of available knowledge, this is the first work simulating a complete Linux-based system with clustered heterogeneous compute cores (in-order and OoO) and an in-cache computing engine, along with 3D stacked HBM2 memory.

In the field of AI and machine learning, gem5-X's compatibility with various machine learning frameworks is a boon for researchers and developers. This feature is instrumental in simulating and evaluating the performance of AI algorithms on different hardware configurations, particularly for hardware accelerators designed for machine learning applications. Additionally, gem5-X's support for external input/output (I/O) models enhances its capability to conduct comprehensive system-level simulations that include various I/O devices and peripherals.

In summary, gem5-X emerges as an indispensable tool in the field of computer architecture simulation. Its comprehensive enhancements in processor and memory modeling, combined with improved simulation efficiency and extensive integration capabilities, render it a vital resource for researchers and developers. Whether it's exploring new processor architectures, digging into the intricacies of memory systems, or integrating complex

hardware and software components, gem5-X provides a robust and versatile platform for advancing the frontiers of computing technology.

2.5 MAESTRO

MAESTRO, short for "Modeling Accelerator Efficiency via Spatio-Temporal Reuse and Occupancy," introduces a comprehensive framework for enhancing the design and optimization of Deep Neural Network (DNN) accelerators. It begins by offering a set of data-centric directives, which provide a succinct and compiler-friendly means to describe DNN dataflow patterns. These directives are instrumental in identifying and harnessing various forms of data reuse, thereby leveraging hardware capabilities efficiently. The effect of different tiling strategies on overall operation, which can be harder to predict, can be estimated with analytical models like MAESTRO. While tiling strategy may have little impact on element-wise operations, it significantly affects operations where data reuse is critical, potentially altering runtime. gem5, compatible with these analytical models, provides an essential simulation environment for understanding such effects in accelerator design.

In the pursuit of achieving both high performance and energy efficiency within DNN accelerators (Figure 6), the optimization of dataflow is paramount. This optimization encompasses how DNN computations are scheduled and mapped across processing elements (PEs). MAESTRO recognizes that effective data reuse, such as maximizing data reuse within PEs and on-chip scratchpads, plays a pivotal role in this context. The energy costs associated with data movement are often more significant than computation costs, underscoring the critical importance of dataflow optimization.

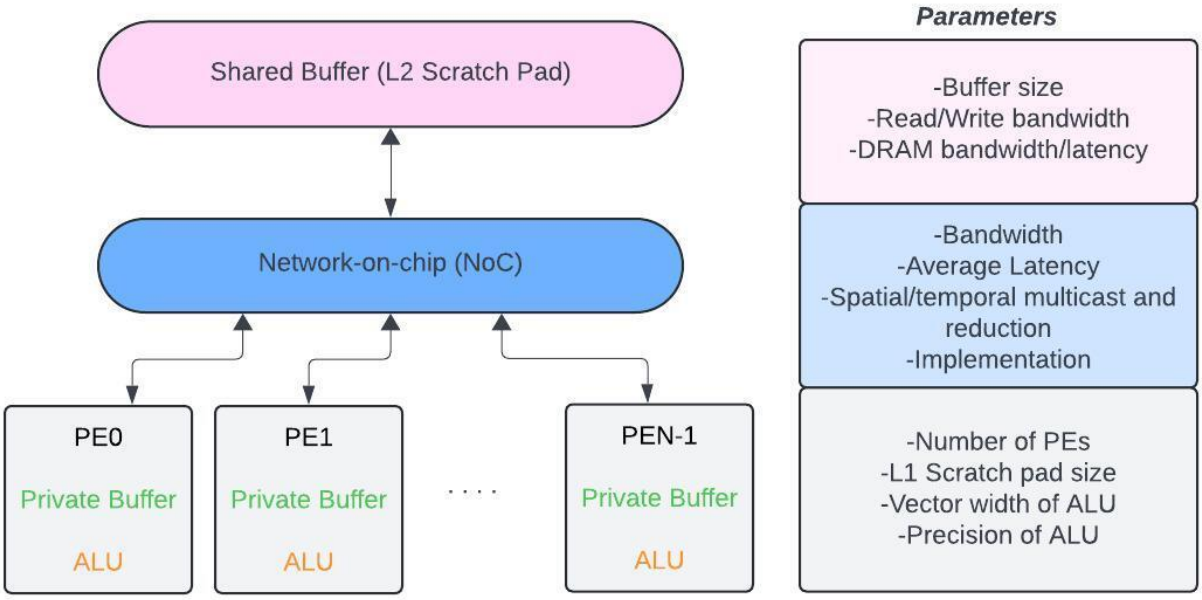


Figure 6: Abstract parameterized DNN accelerator

MAESTRO introduces an analytical cost model that explores the intricate trade-offs inherent in dataflow design, particularly in the context of neural network (NN) architectures. NN architectures, which are fundamental to deep learning, consist of interconnected layers of nodes or neurons that process data through various transformations. These architectures can vary significantly in their structure, such as feedforward networks, convolutional neural networks (CNNs), and recurrent neural networks (RNNs), each having unique characteristics and computational requirements.

Unlike simulation-based approaches, MAESTRO does not rely on executing simulations to evaluate different design choices. Instead, it uses mathematical models to predict the performance, energy consumption, and resource utilization of different dataflow mappings on hardware accelerators [17]. This distinction is crucial for several reasons, including speed and scalability, insight and understanding, predictive capability, and focus on dataflow [13]. MAESTRO is a powerful tool that estimates various aspects, including execution time and energy efficiency, concerning both the DNN model and the hardware configuration. This means it analyzes how different designs of neural networks interact with the underlying hardware, optimizing for both the architecture of the neural network and the capabilities of the hardware, such as GPUs or TPUs.

Furthermore, the framework emphasizes the synergistic optimization of both hardware microarchitecture and the associated dataflows, which is a crucial objective for accelerator design. The term "dataflow" refers to the movement of data through the computational graph

of a neural network. Different dataflow strategies (e.g., streaming, batch processing, or pipelining) can significantly affect performance and efficiency. The microarchitecture, which includes aspects like memory hierarchy and processing units, influences how data is handled. MAESTRO explores how different ways of organizing dataflow can impact the hardware's performance, ultimately leading to more efficient designs.

By providing a structured mechanism for evaluating dataflow choices and their interplay with microarchitectural alternatives, MAESTRO offers a thorough understanding of these complex relationships. Key contributions include a data-centric notation for representing accelerator dataflows, facilitating insight into data mappings and reuses. Moreover, it introduces three transformations that encompass the complete spectrum of dataflow possibilities:

- **Scheduling**: This involves determining the order in which operations (like matrix multiplications or activations) are performed. Effective scheduling can minimize waiting times and improve resource utilization, ensuring that the processing units are kept busy.
- **Tiling**: Tiling breaks down large data sets or operations into smaller, manageable pieces or "tiles." This technique helps improve data locality, allowing the hardware to access data more efficiently and reducing memory access times, which is critical for performance in large DNNs.
- **Mapping**: Mapping refers to the assignment of specific computations or data to particular hardware resources (like cores or memory banks). By strategically mapping operations to the right resources, MAESTRO can optimize performance, leveraging the unique strengths of the hardware.

MAESTRO's significance lies not only in its ability to identify data reuse opportunities but also in its practicality. It acknowledges that appropriate hardware support is essential for realizing these opportunities. Hardware communication types, such as multicast for input tensors and reduction for output tensors, are crucial considerations. For instance, multicast delivers the same data to multiple targets in either space or time, demanding specific network-on-chip structures or stationary buffers. Reduction, on the other hand, processes partial sums to generate final outputs.

MAESTRO aids in understanding the trade-offs between computational cost and performance. This understanding is essential in making informed decisions about the complexity and cost-effectiveness of the hardware design. For instance, a more complex dataflow strategy might offer better performance but could also lead to increased power consumption and chip area. MAESTRO provides the necessary analytics to balance these aspects effectively. Dataflow, in this context, refers to the pattern in which data is passed through the computational units of the accelerator [43][44]. Different dataflow strategies can lead to significant variations in performance, power consumption, resource utilization and neural network accuracy.

Moreover, MAESTRO allows designers to simulate and analyze these different strategies, providing a comprehensive view of their impact on the accelerator's efficiency. It offers insights into the optimal use of resources such as memory bandwidth and computational units, helping in designing accelerators that are both high-performance and energy-efficient. This aspect of MAESTRO is especially important in scenarios where power and resource constraints are critical, such as in mobile and edge computing devices.

One of the key applications of MAESTRO is in the design of neural network accelerators. These accelerators, which are specialized hardware designed to efficiently run neural network models, can greatly benefit from MAESTRO's analysis. For example, in designing accelerators for Convolutional Neural Networks (CNNs), MAESTRO can model different dataflow strategies and predict their impact on throughput and efficiency. This predictive capability was particularly relevant in the design of specialized hardware like Google's TPU (Tensor Processing Unit), which is optimized for tensor operations commonly found in neural networks.

In the realm of edge computing, where devices often operate under stringent power constraints, MAESTRO's role becomes even more critical. It assists in creating accelerators that are optimized for power efficiency while still delivering the necessary computational power. This is vital for applications like autonomous vehicles and Internet of Things (IoT) devices, where processing needs to be fast yet power-efficient.

In high-performance computing (HPC) environments, where processing large and complex datasets swiftly is a priority, MAESTRO contributes by aiding in the design of accelerators capable of handling such demanding workloads efficiently. It helps in reducing computation time and energy consumption, which are key considerations in HPC systems.

In summary, MAESTRO is indispensable for deep learning accelerator design, offering predictive analytics, resource analysis, and optimization capabilities crucial for advancing AI technologies.

2.6 Similarities

At their core, all these tools aim to advance SoC design and simulation. They are pivotal in analyzing and optimizing performance, energy efficiency, and resource utilization, which are critical aspects in the design of efficient and effective computing systems. They offer detailed simulation and modeling capabilities, enabling designers and researchers to gain in-depth insights into complex computing architectures. This common ground in functionality underscores their shared purpose in the field of computer architecture.

While these technologies share a common goal of advancing SoC simulation, their distinct features and focus areas set them apart from each other in significant ways:

gem5-Aladdin vs. SMAUG:

gem5-Aladdin: Tailored for co-simulation of CPUs and hardware accelerators, gem5-Aladdin excels in analyzing performance across various architectural designs and configurations. It integrates the gem5 simulator with Aladdin, focusing on the detailed simulation of SoC architectures.

SMAUG: Offers a broader platform for heterogeneous system simulation. While also supporting CPUs and accelerators, SMAUG is recognized for its versatility and wider applicability in simulating diverse systems and especially DNN.

gem5-SALAM vs. MAESTRO:

gem5-SALAM: An extension of the gem5 simulator, gem5-SALAM specializes in co-simulating CPU components with advanced accelerators. It emphasizes system-level analysis, examining the interplay between different SoC components.

MAESTRO: Concentrates on dataflow efficiency within hardware accelerators. Its focus is on optimizing dataflow strategies, essential for high-efficiency accelerator design.

MAESTRO vs. SMAUG:

MAESTRO: Primarily focused on optimizing dataflow within specialized accelerators. It is highly effective for designs where dataflow efficiency is a key consideration.

SMAUG: More versatile in its application, SMAUG can handle a range of simulation scenarios, making it a go-to for diverse system simulations, including those with complex architectural requirements.

gem5-SALAM vs. gem5-Aladdin:

gem5-SALAM: This framework extends gem5's capabilities to co-simulate advanced accelerators alongside CPUs, emphasizing comprehensive system-level analysis.

gem5-Aladdin: Specifically designed for detailed simulations involving CPU and accelerator co-operation, it is adept at exploring the performance impacts of different SoC configurations.

gem5-X vs. Others:

gem5-X: Distinguishes itself with broad enhancements to the original gem5 simulator. Its versatility extends to improved processor and memory modeling, along with enhanced simulation speed and efficiency. gem5-X is adaptable for a variety of simulation tasks, making it a versatile tool in SoC design.

In essence, while all these tools are geared towards advancing SoC design and simulation, their differences lie in their specific areas of specialization. gem5-Aladdin and SMAUG are more focused on the interaction between CPUs and accelerators, with the former providing a more specialized environment for this purpose, while the latter offers greater general applicability. gem5-SALAM and MAESTRO both explore the accelerator design, but from different angles – gem5-SALAM from a system-level perspective and MAESTRO from the standpoint of internal dataflow efficiency. gem5-X, meanwhile, serves as a more comprehensive extension of the gem5 simulator, enhancing its overall capabilities across a broader spectrum of architectural simulation tasks.

This comparative analysis underscores the importance of selecting the appropriate tool based on the specific requirements of a given simulation or design task, highlighting the specialized nature of these advanced SoC design and simulation frameworks.

2.7 Architectural Details of Each Framework

2.7.1 Framework Architectures

Focusing solely on the architectural analysis of the gem5-Aladdin framework:

- Co-Design Approach: The architectural foundation of gem5-Aladdin is built on the principle of co-designing accelerator microarchitectures with their respective systems. This approach is crucial for creating balanced and efficient designs, especially given the complexity and diversity of hardware accelerators in modern SoCs. The framework emphasizes the importance of considering data movement and coherence management, which are often overlooked but significant in determining the overall performance and efficiency of accelerators.
- Dynamic Interaction Simulation: At its core, gem5-Aladdin is designed to simulate the dynamic interactions between accelerators and the broader SoC platform. This aspect of the architecture is critical for accurately predicting the performance of accelerators in real-world scenarios. By capturing these interactions, gem5-Aladdin can provide more realistic and useful insights into the design and optimization of accelerator microarchitectures.
- Aladdin as a Power-Performance Simulator: Aladdin, a key component of the framework, functions as a pre-RTL power-performance simulator. It is architecturally distinct in that it uses dynamic data dependence graphs (DDDG) to represent accelerators based on high-level language descriptions of algorithms. This eliminates the need for low-level RTL generation, streamlining the design process. The architectural design of Aladdin allows for the application of optimizations and constraints to the DDDG, resulting in a realistic model of accelerator activity that is validated for performance, power, and area.
- Integration Challenges and Solutions: The architecture of gem5-Aladdin addresses the challenges in current SoC designs, which often lack higher-level coordination and

optimization between various components such as general-purpose cores, accelerators, and shared resources. The framework advocates for a design methodology that facilitates broader design space exploration of customized architectures, considering these elements in an integrated manner.

- In essence, the architectural analysis of the gem5-Aladdin framework reveals a sophisticated and forward-thinking approach to SoC design. It underscores the importance of co-design and system-level consideration in developing efficient and effective hardware accelerators, marking a significant advancement in the field of computer architecture.

The SMAUG framework represents a significant advancement in the field of deep learning (DL) research, particularly in hardware acceleration for deep neural networks (DNNs). Its architecture is designed to address the challenges in studying end-to-end DNN performance during early-stage design, especially before Register-Transfer Level (RTL) development. SMAUG stands out for its capability to simulate deep learning applications in a full-stack manner, from hardware accelerators to software frameworks.

As of its keys features:

- Integration with gem5-Aladdin: SMAUG is closely integrated with gem5-Aladdin, a System on Chip simulator. This integration supports the modeling of complex heterogeneous SoCs, making SMAUG the first architecture-simulation friendly deep learning framework. This design enables rapid evaluation of different accelerator and SoC designs, and facilitates hardware-software co-design.
- Python API and C++ Runtime: The framework is divided into three major components:
    - A Python frontend for network configuration, which allows users to build networks using a declarative style. This part handles input and weights data, and the configuration of accelerated kernels.
    - A C++ runtime manages the execution flow, handling the operation of the deep learning models.
    - A backend consisting of hardware-accelerated kernels, which can be modeled using Aladdin or as native gem5 simulation objects.
- Tiling Optimizer and Scheduler: SMAUG features a tiling optimizer that computes the best tiling shapes for each operation, ensuring optimal use of the accelerator's compute and memory resources. The scheduler then prepares and dispatches these tiles to the appropriate compute elements, managing multiple accelerators and threads efficiently.
- Flexible Backend Implementation: The backend supports a range of operations required by DNN models, including convolutions and inner products. The models for these operations can be written using Aladdin or as native gem5 objects. This flexibility allows users to implement new hardware accelerator models easily into the framework.

- Sampling Support with Aladdin: SMAUG extends Aladdin to include a new API for sampling at the per-loop level. This feature allows for efficient simulation of DNNs, reducing the time required for forward pass simulation significantly while maintaining accuracy.
- Efficient Operation within Simulator Limitations: SMAUG is designed to operate effectively within the limitations of user-space simulators. It minimizes its interactions with the operating system, running as a single C++ binary and managing its thread operations within these constraints.
- Power and Area Modeling: The framework also includes capabilities for power and area estimation. This is done by characterizing various 16-bit functional units and modeling accelerator local scratchpads, among other components.
- Case Studies and Performance Optimization: SMAUG demonstrates its utility through various case studies. These studies show how the framework can optimize data transfers, exploit tile-level parallelism in multi-accelerator systems, and improve tiling transformations in software. These optimizations significantly speed up overall inference latency.

The architecture of SMAUG is carefully designed to bridge gaps in deep learning research infrastructure, particularly in the early stages of hardware-software co-design. Its integration with gem5-Aladdin, Python API, tiling optimizer, flexible backend, and efficient operation within simulator limitations collectively make it a powerful tool for DNN researchers. The framework's ability to simulate and optimize end-to-end DNN performance makes it a significant contribution to the field of deep learning and hardware acceleration.

The MAESTRO framework is a highly specialized tool designed for understanding and optimizing dataflows in deep neural network (DNN) accelerators. Its architecture and implications can be summarized as follows:

- Data-Centric Approach: MAESTRO employs a data-centric approach for representing various accelerator dataflows, emphasizing data mappings and reuses as primary elements. This contrasts with traditional compute-centric notations, enabling a more precise analysis of data reuse and movements across the accelerators.
- Analytical Cost Model: It introduces an analytical cost model named MAESTRO (Modeling Accelerator Efficiency via Spatio-Temporal Reuse and Occupancy) for estimating the cost-benefit tradeoffs of a dataflow, including execution time and energy efficiency. This model is essential for evaluating different design choices in terms of performance and energy usage.
- Dataflow Optimization: The framework focuses on optimizing dataflow for DNN accelerators. The performance and energy efficiency of these accelerators heavily depend on how the computations are scheduled and mapped across processing elements (PEs). MAESTRO helps in understanding and optimizing these aspects for improved efficiency.

- Reuse Taxonomy and Hardware Implementation: MAESTRO identifies and utilizes various data reuse opportunities, like multicasting and reduction, to enhance the efficiency of DNN accelerators. It also provides insights into suitable hardware implementations for these reuse strategies, helping in the design of more efficient accelerators.
- Key Directives and Clustering: The architecture involves key directives like spatial and temporal mapping, data movement order, and clustering. These directives help in organizing data flows and processing across the accelerator, impacting how effectively the DNN computations are executed.
- Dataflow Transformations: MAESTRO encompasses transformations such as scheduling, tiling, and mapping, providing a comprehensive framework for understanding all aspects of dataflows in DNN accelerators.

In summary, MAESTRO provides a sophisticated framework for DNN accelerators, focusing on data-centric analysis and optimization of dataflows, which is crucial for achieving high performance and energy efficiency in DNN computations.

One of the key features of gem5-SALAM is its "execute-in-execute" LLVM-based model. This model is distinct from traditional simulation methods in that it allows for a more integrated and efficient way of simulating hardware accelerators. The use of LLVM (Low Level Virtual Machine) as the basis for modeling ensures that gem5-SALAM is adaptable and capable of simulating a wide range of hardware accelerators in a more realistic environment. This adaptability is crucial for modern system architectures, which are increasingly reliant on custom hardware accelerators to meet the demands of advanced computational tasks.

The architecture of gem5-SALAM necessitates specific requirements, including dependencies on gem5 and LLVM-9 or newer versions. The setup involves the use of a front-end LLVM compiler for the preferred development language, like clang for C, which facilitates the integration of the model with the gem5 environment. This integration is a critical aspect of the framework, as it enables the seamless interaction between the simulated hardware accelerators and the rest of the system modeled in gem5.

In the gem5-SALAM framework, accelerators are constructed by associating their LLVM Intermediate Representation (IR) with an LLVMInterface. This interface is then connected to the desired CommInterface in the gem5 memory map. This approach underscores the framework's flexibility and its ability to cater to a wide range of system configurations and requirements.

This framework represents a significant step forward in the field of system architecture and hardware accelerator modeling, offering a more versatile and efficient approach to simulation that can adapt to the rapidly evolving demands of modern computing systems.

The Top accelerator within the gem5-SALAM framework is an essential component designed to manage and control the operations of specialized hardware accelerators. It functions as the primary interface between the system's main processing unit and the

specialized hardware, orchestrating data flow and control signals to ensure efficient and synchronized operations.

In the context of the gem5-SALAM framework, the Top accelerator's role is particularly pivotal in managing hardware components designed for specific tasks. It's tasked with initializing and controlling these hardware accelerators, as well as managing the Direct Memory Access (DMA) operations that are crucial for the efficient transfer of data between the system's main memory and the accelerators' scratch-pad memory. This capability is critical for operations that require rapid and frequent data exchanges, such as complex computational tasks or data-intensive processes.

The operational mechanism of the Top accelerator involves receiving and interpreting commands from the system's CPU, initiating the appropriate hardware accelerators, and overseeing the DMA processes. The control mechanism is implemented through a series of Memory Mapped Registers (MMRs), which the CPU utilizes to communicate with the Top accelerator. These MMRs serve as a bridge, allowing the CPU to pass control signals and data addresses to the Top accelerator, which then translates these into actions for the hardware accelerators and DMAs.

The Top accelerator's code, typically organized in a file like top.c, begins with the declaration of these MMR addresses. These addresses correspond to locations in the accelerator's memory space, which the CPU later fills with relevant data. Additionally, the Top accelerator sets up static addresses linked to the MMRs of the connected hardware accelerators, enabling direct control over these components.

A significant feature of the Top accelerator is its ability to manage data transfers through DMAs. This process involves setting up the MMRs of the DMA to execute memory transfers efficiently between the main memory and the accelerator's scratch-pad memory. The Top accelerator initiates these transfers and continuously polls the DMA for completion status, ensuring a seamless data flow crucial for the optimal functioning of the hardware accelerators.

The Top accelerator's capabilities are not just limited to managing data flow but also extend to optimizing system performance. By efficiently controlling the hardware accelerators and DMAs, the Top accelerator plays a critical role in reducing the computational load on the main CPU, allowing for more efficient overall system performance. This is particularly important in scenarios where the system is running complex, data-intensive tasks that would otherwise significantly burden the main processor.

The gem5-X framework is an advanced full-system simulator based on the gem5 simulator, with a focus on architectural extensions and the optimization of heterogeneous systems. Key aspects of the gem5-X framework include:

- Architectural Exploration and Optimization: gem5-X facilitates architectural exploration and optimization, particularly for heterogeneous systems. It integrates advanced features like in-cache computing and 3D stacked High Bandwidth Memory, allowing for a detailed analysis of system performance and potential bottlenecks.

- Performance Validation: The framework's performance results have been validated against real-world hardware, such as the ARMv8 JUNO board, demonstrating a high accuracy rate (below 4% error) in execution time comparisons.
- Support for State-of-the-Art Processors: gem5-X is designed to support current leading-edge processors, including ARMv8 in-order and out-of-order architectures. This makes it a valuable tool for exploring and optimizing modern processing architectures.
- User-Friendly Environment: Aimed at computer architects from both academia and industry, gem5-X offers a user-friendly environment that simplifies the learning curve associated with gem5. It provides a fully operational environment and setup right out of the box, fostering easier and faster development.
- Open-Source and Community Engagement: As an open-source project, gem5-X encourages engagement and contribution from the wider computer architecture community. This collaborative approach enhances the framework's capabilities and aligns it closely with real-world application requirements and architectural constraints.
- Specialized Architectural Extensions: The framework includes specialized environments for various architectural extensions, such as Analog In-Memory Cores (AIMCs), wireless in-package links, and tightly-coupled systolic arrays. These extensions are available in separate repositories within the gem5-X GitHub organization, underscoring the framework's versatility and adaptability to diverse architectural needs.

Overall, gem5-X represents a significant development in full-system simulation, offering comprehensive tools for analyzing and optimizing complex and heterogeneous system architectures.

2.7.2 Design Philosophies

In the intricate field of computer architecture and hardware acceleration, the design of frameworks like gem5-Aladdin, SMAUG, gem5-SALAM, MAESTRO, and gem5-X is not just a matter of technical necessity but also a reflection of visionary foresight. These frameworks are conceived to address the multifaceted challenges and leverage the opportunities presented by the dynamic landscape of modern computing.

gem5-Aladdin stands as a prime example of innovative design in the realm of hardware-software co-simulation. Its philosophy is anchored in the integration of system-level and accelerator-level simulations, providing a dual lens through which the interplay of hardware accelerators with traditional CPU and memory systems can be studied. The rationale behind this is rooted in the recognition that the future of computing lies in the synergy between heterogeneous computing elements. The ability of gem5-Aladdin to simulate diverse accelerator configurations alongside CPUs offers invaluable insights into performance bottlenecks and energy efficiency, making it an indispensable tool in the era of specialized computing.

SMAUG takes a slightly different approach, focusing on the burgeoning field of neural network accelerators. Its design philosophy emphasizes adaptability and user-friendliness, recognizing the rapid pace at which this field is evolving. By providing a flexible platform that

can accommodate a wide array of accelerator architectures, SMAUG serves as a crucible for innovation, enabling researchers and developers to experiment and iterate rapidly. This design philosophy is driven by the understanding that the future of neural network computing is one of continuous evolution and adaptation, and SMAUG positions itself as a versatile tool in this ever-changing landscape.

Moving to gem5-SALAM, this framework builds upon the gem5 architecture to offer detailed insights into the interaction between advanced technologies and microarchitectural design. Its design philosophy is centered around the accuracy of simulation, recognizing the critical role that detailed architectural exploration plays in the advancement of autonomous systems and other high-tech applications. The rationale is that as these technologies become increasingly integrated into our computing infrastructure, understanding their impact on microarchitectural design and system performance becomes essential. gem5-SALAM's ability to simulate these interactions in a detailed and holistic manner makes it a pivotal tool for researchers aiming to push the boundaries of advanced hardware development.

MAESTRO stands out with its focus on the efficiency of deep learning accelerators. Its design philosophy is deeply analytical, emphasizing the importance of modeling dataflow and resource allocation in hardware designs. The rationale behind MAESTRO is the complexity of resource management in deep learning accelerators, where efficient utilization of spatio-temporal resources is key to maximizing performance. By providing a framework that can dissect and analyze these aspects, MAESTRO aids in the optimization and advancement of deep learning hardware designs, a critical area in the age of AI-driven computing.

Lastly, gem5-X is tailored for the simulation of many-core systems, a field that is rapidly gaining prominence. Its design philosophy revolves around comprehensive simulation, recognizing the unique demands posed by many-core technologies. The rationale for this focus is the growing importance of many-core systems in various sectors, necessitating a tool that can accurately simulate and analyze the complex interplay of hardware and software in these systems. gem5-X’s specialization in many-core technologies positions it as a crucial tool in understanding and optimizing the performance of these emerging systems.

In summary, the design philosophies and rationales of these frameworks reflect an in-depth understanding of the current and future landscape of computer architecture and hardware acceleration. From the integrated approach of gem5-Aladdin to the efficiency-focused modeling of MAESTRO and the many core-centric simulation of gem5-X, these frameworks are pivotal in guiding the direction of modern computing. Their development is not just a response to current technological needs but a proactive step towards shaping the future of computing, emphasizing the need for flexibility, accuracy, and specialization in addressing the complex and evolving challenges of this field.

2.8 Operational Mechanisms and Workflow of the Frameworks

In the dynamic field of computer architecture and hardware acceleration, frameworks like gem5-Aladdin, SMAUG, gem5-SALAM, MAESTRO, and gem5-X stand out for their specialized workflows, operational mechanisms, and practical applications in various use case scenarios. gem5-Aladdin exemplifies innovation in hardware-software co-simulation. Its workflow integrates the system-level simulation capabilities of gem5 with Aladdin's hardware accelerator simulation, enabling simultaneous evaluation of CPU and accelerator designs. This seamless integration facilitates a dual lens approach to studying the interplay of hardware accelerators with traditional CPU and memory systems. Operationally, gem5-Aladdin thrives on co-simulation, where data exchange and synchronization between gem5 and Aladdin ensure accurate system representation. A notable use case is in energy-efficient accelerator design, where researchers can design, test, and refine accelerators for scenarios demanding low-power consumption, such as in mobile devices. SMAUG, with its focus on neural network accelerators, adopts a workflow that emphasizes flexibility and adaptability. Its core simulation engine is designed to accommodate a range of accelerator architectures, fostering an environment conducive to iterative design and rapid prototyping. SMAUG's modular design and performance analysis tools allow for detailed evaluation of accelerator efficiency and throughput, making it ideal for optimizing neural network accelerators for specific tasks like image recognition. In gem5-SALAM, the workflow is centered around the co-simulation of system-level and specialized accelerator components, with a strong emphasis on detailed architectural analysis. This framework enhances the capabilities of gem5 to include simulations of specialized accelerators, capturing their complex interactions with traditional computing elements. Its operational strength is in providing a system-wide performance analysis, which is crucial for understanding the impact of advanced technologies on microarchitectural design. A practical application of gem5-SALAM is in optimizing hardware for autonomous systems, where system-level performance is of utmost importance. MAESTRO stands out for its analytical approach to deep learning accelerator efficiency. The workflow is centered around modeling dataflow and resource allocation, employing predictive models to understand and enhance accelerator performance. MAESTRO's analytical prowess is particularly evident in its resource utilization analysis, which is key to maximizing performance in deep learning tasks. This framework is particularly beneficial in the design and optimization of deep learning accelerators for processing large datasets in applications like speech recognition and computer vision. Finally, gem5-X specializes in the comprehensive simulation of many-core system platforms. Its workflow is tailored to provide a detailed evaluation of many-core system components, including CPUs and GPUs. gem5-X operates by co-simulating hardware and software aspects of many-core systems, identifying performance bottlenecks and guiding optimizations for improved user experiences. This makes gem5-X invaluable for developers focusing on applications such as immersive gaming or virtual reality environments, where high-performance and responsive systems are crucial. Collectively, these frameworks represent a spectrum of tools designed to address the evolving needs of modern computing. From the energy-efficient design possibilities with gem5-Aladdin to the immersive many-core system optimizations with gem5-X, each framework brings unique capabilities to the table. Their workflows and operational mechanisms are finely tuned to their respective focus

areas, providing researchers and developers with powerful tools for simulation, analysis, and optimization in the complex world of computer architecture and hardware acceleration.

2.9 Advantages and Disadvantages of Each Framework

Analyzing the advantages and disadvantages of frameworks like gem5-Aladdin, SMAUG, gem5-SALAM, MAESTRO, and gem5-X involves a comprehensive assessment of their strengths and limitations, as well as a comparative analysis to understand their relative efficacy in various scenarios.

gem5-Aladdin's biggest strength lies in its ability to integrate CPU and hardware accelerator simulations, providing a holistic view of system performance. Moreover, it enables detailed analysis of power consumption and performance, key for optimizing energy efficiency in hardware designs. Finally, it supports a wide range of accelerator models, offering flexibility in hardware design and experimentation. On the other hand, it has some limitations such as the integration of two complex systems (gem5 and Aladdin) can lead to a steep learning curve and complexity in setup and usage. Furthermore, the detailed simulations can be resource-intensive, requiring significant computational power. Compared to other frameworks, gem5-Aladdin stands out for its integrated approach to CPU and accelerator simulation. However, its complexity and resource requirements may be challenging for some users, especially when compared to more focused frameworks like SMAUG or MAESTRO.

SMAUG is specialized in simulating neural network accelerators, making it highly relevant in the AI and ML fields. Its design emphasizes user-friendliness and adaptability, allowing for rapid prototyping and iteration. Moreover, it offers detailed tools for analyzing accelerator efficiency and throughput. On the other hand, it is specialized focus on neural network accelerators can be a limitation for users interested in a broader range of hardware simulations. Unlike gem5-Aladdin, SMAUG may lack comprehensive system-level simulation capabilities. SMAUG's specialized focus on neural network accelerators makes it uniquely suited for AI and ML applications, but less versatile than gem5-Aladdin for general hardware-software co-simulation. Its user-friendliness is a distinct advantage over more complex frameworks like gem5-SALAM.

gem5-SALAM excels in simulating the intricacies of specialized accelerators within computing systems. It provides a comprehensive analysis of how advanced technologies impact overall system performance. The framework's focus on microarchitectural design is crucial for optimizing the integration of advanced technologies. Setting up and configuring detailed simulations can be complex and time-consuming. Like gem5-Aladdin, gem5-SALAM's simulations can be quite resource-intensive. gem5-SALAM’s detailed focus on accelerators and microarchitectural impacts offers a more specialized perspective compared to the broader approach of gem5-Aladdin. However, this specialization comes at the cost of increased complexity and resource requirements.

MAESTRO specializes in modeling the efficiency of deep learning accelerators, an area of growing importance. Its analytical tools provide valuable insights into dataflow and resource allocation, aiding in the optimization process. The emphasis on resource utilization is key for designing high-performance deep learning hardware. MAESTRO's niche focus on deep learning accelerator efficiency might limit its applicability for broader hardware simulations. Understanding and effectively using its analytical models may require a significant learning curve. MAESTRO's specialized focus on deep learning accelerator efficiency sets it apart from frameworks like gem5-Aladdin and SMAUG, which offer broader hardware simulation capabilities. Its analytical depth is a significant strength, though it may be more challenging to master for new users.

gem5-X is uniquely focused on the simulation of many-core system platforms, catering to a rapidly growing technological field. Co-Simulation: It offers a detailed simulation environment for both hardware and software components of many-core systems. The framework is well-suited for identifying and addressing performance bottlenecks in many-core applications. High-fidelity simulation of many-core systems can be resource-demanding. gem5-X's specialization in many-core systems fills a unique niche compared to other frameworks. While it offers detailed insights into many-core system performance, its applicability is more limited compared to the broader simulation capabilities of frameworks like gem5-Aladdin or SMAUG.

In our exploration of the gem5-X framework, an essential resource has been the official gem5-X Technical Manual, provided by EPFL. While this manual serves as a primary guide for understanding and working with gem5-X, our experience has revealed certain limitations in its comprehensiveness and practical applicability, particularly for advanced users and complex simulation scenarios.

One notable aspect is the relative simplicity of the examples provided in the manual. While these examples are undoubtedly useful for beginners to get started with gem5-X, they fall short in demonstrating the full spectrum of the framework's capabilities. For researchers and developers who are looking to leverage gem5-X for more sophisticated System-on-Chip simulations, especially those involving intricate CPU-GPU interactions or advanced architectural configurations, the manual's examples do not suffice. They lack the depth and complexity needed to fully grasp the potential applications and intricacies of gem5-X in high-end SoC design and simulation tasks.

Moreover, the thesis encountered specific challenges when attempting to implement the provided examples on Out-of-Order (OoO) CPU architectures. These challenges were not adequately addressed in the documentation, leading to a trial-and-error approach in our attempts to run these simulations. The lack of detailed guidance or troubleshooting tips for running examples on OoO CPUs points to a gap in the manual, which could be critical for users who are working with these types of advanced CPU architectures.

These observations suggest a need for a more comprehensive and detailed technical manual for gem5-X, one that encompasses a wider range of examples and scenarios, including those pertinent to advanced users. Additionally, more thorough documentation on

handling simulations on OoO CPUs would significantly enhance the utility of the manual, enabling users to better navigate and resolve issues specific to these complex architectures.

In conclusion, while the gem5-X Technical Manual is a valuable starting point for engaging with the framework, its limitations in scope and detail, particularly for advanced simulations and specific CPU architectures, highlight an area for improvement. Enhancing the manual with more complex examples and detailed guidance for OoO CPU simulations would greatly benefit the gem5-X user community, fostering a deeper understanding and more effective utilization of this powerful simulation framework.

Overall Comparative Perspective

Each framework has its unique strengths and limitations, shaped by its focus and design philosophy. gem5-Aladdin and gem5-SALAM offer broad system-level simulation capabilities, but are more complex and resource-intensive. SMAUG and MAESTRO, with their specialized focus on neural network and deep learning accelerators, respectively, offer depth in their areas but at the expense of broader applicability. gem5-X, with its unique focus on many-core systems, caters to a niche yet growing field. In summary, the choice of framework largely depends on the specific needs and focus of the user. For broad system-level simulations, gem5-Aladdin and gem5-SALAM are more suited, while SMAUG and MAESTRO are ideal for specific accelerator design and optimization tasks.

2.10 Advantages, disadvantages, and critical aspects

The journey through the earlier chapters of this dissertation has meticulously unfolded the multifaceted world of System-on-Chip design and simulation frameworks. The thesis has delved deep into the design philosophies, architectural details, and operational intricacies of frameworks such as gem5-Aladdin, SMAUG, gem5-SALAM, MAESTRO, and gem5-X. This section, building on the rich analysis presented earlier, introduces a comparative table that synthesizes and contrasts these frameworks in a comprehensive manner. The depth of our exploration is particularly evident in the intricate details of gem5-SALAM. From the modular architecture of its components, such as the CommInterface and LLVMInterface, to the specialized features of its memory and port systems, these insights are seamlessly woven into our comparative narrative. This table serves not just as a summary, but as an analytical tool, bridging the complex technical discussions with a structured and accessible format. It thereby enhances the reader's ability to make informed decisions in the dynamic field of SoC simulation.

Table 1: Aspects of each framework

Framework Name	Simulation Accuracy	Simulation Speed	Flexibility	Hardware Support	Software Interface	Ease of Use
gem5-Aladdin	High	Moderate	High	Extensive	User-friendly	Moderate
SMAUG	Moderate	High	Moderate	Limited	Complex	Low
gem5-SALAM	High	High	Very High	Extensive	Moderate	High
MAESTRO	High	Moderate	High	Moderate	User-friendly	High
gem5-X	Moderate	High	High	Extensive	Complex	Moderate

Table 2: More features and characteristics

Framework Name	Year of Presentation	Integration	Community and Development Activity	Ideal Use-Cases	Performane Metrics
gem5-Aladdin	Jun 2016	Easy	*Support*: Strong *Last Update*: Jan 2022	Academic Research, CPU Co-Simulation	Fast, Accurate
SMAUG	Nov 2020	Moderate	*Support*: Moderate *Last Update*: Oct 2021	Industrial Use, Deep Learning Applications	Scalable, Efficient
gem5-SALAM	Oct 2020	Easy	*Support*: Strong *Last Update*: Feb 2024	Academic Research, Heterogeneous System Simulation	Very Fast, Accurate
MAESTRO	Oct 2019	Moderate	*Support*: Strong *Last Update*: Aug 2022	Dataflow Architecture Simulation, Academic Research	Accurate, Resource-Efficient
gem5-X	Apr 2019	Difficult	*Support*: Moderate *Last Update*: Aug 2021	Industrial Use, High-Performance Computing	Efficient, Scalable

The table begins with the 'Framework Name', which lists the SoC simulation frameworks, including gem5-SALAM, renowned for its modular components like CommInterface, LLVMInterface, and various specialized ports and memory systems:

- 'Simulation Accuracy' gauges how precisely each framework, such as gem5-SALAM with its LLVMInterface, simulates SoC designs. This precision directly impacts the reliability and applicability of the framework in replicating real-world scenarios.
- In 'Simulation Speed', the efficiency of frameworks is evaluated. For instance, gem5-SALAM's efficient memory-to-memory transfers facilitated by components like NoncoherentDma and StreamDma are indicative of its operational efficiency.
- 'Flexibility' measures each framework's adaptability. The modular design of gem5-SALAM, featuring various ports and interfaces, highlights its versatility in handling diverse simulation requirements.
- The 'Hardware Support' column looks at the range of hardware each framework can simulate. Gem5-SALAM’s ScratchpadMemory and StreamBuffer demonstrate its capability to support custom, fast-access memory and facilitate efficient data transfers.
- 'Software Interface' assesses the usability of the frameworks. The detailed and functional interfaces of gem5-SALAM, as evident from its CommInterface and other components, play a crucial role in user interaction and ease of operation.
- 'Ease of Use' reflects on the overall user experience. The structured layout and clear documentation of frameworks like gem5-SALAM enhance user accessibility and operational efficiency.
- 'Integration with Other Systems' evaluates how well each framework integrates into broader ecosystems, a trait exemplified by gem5-SALAM’s AccCluster, which organizes accelerators and shared resources.
- 'Community and Development Activity' considers the level of community engagement, the year of presentation, and the last update of the code in the GitHub repository to assess the project's current viability. Frameworks with robust components and clear documentation, such as gem5-SALAM, typically benefit from strong community involvement and regular updates.
- 'Ideal Use-Case Scenarios' identifies the best applications for each framework, with gem5-SALAM's diverse components making it suitable for a range of scenarios, from academic research to industrial applications.
- 'Performance Metrics' quantitatively compare each framework’s capabilities. For example, gem5-SALAM’s StreamDma and ScratchpadMemory indicate its efficiency and scalability in a system on chip.
- 'Future Compatibility' looks at the potential for each framework to evolve with technological advancements. The modular and adaptable nature of frameworks like gem5-SALAM suggests their readiness for future developments.
- 'Additional Notes' allows for highlighting unique features or observations specific to each framework, such as gem5-SALAM’s specialized memory systems and communication protocols.

This comprehensive table, enriched with specific details about frameworks like gem5-SALAM, serves as a pivotal resource in the field of SoC design and simulation. It not only

encapsulates the key features, strengths, and weaknesses of various frameworks but also reveals crucial insights for the selection process. For instance, the table highlights gem5-Aladdin and gem5-SALAM for their high simulation accuracy and flexibility, making them ideal for academic research and heterogeneous system simulation. Meanwhile, SMAUG stands out for its high simulation speed and efficiency, suited for industrial use and deep learning applications. MAESTRO, with its balanced attributes, is well-suited for dataflow architecture simulation. The table also sheds light on future compatibility, indicating that frameworks like gem5-SALAM are well-equipped to adapt to future technological advancements, a critical consideration in this rapidly evolving field. Such distinctions are invaluable for researchers and developers, guiding them towards frameworks that best align with their specific needs, whether for in-depth research, practical applications, or innovative, future-oriented development. This comparative analysis, therefore, stands not just as a testament to the intricate and dynamic nature of SoC simulation frameworks, but also as a practical guide for informed decision-making in the realm of SoC design and simulation.

Table 3: ISA and usage description of each framework

Framework Name	ISA Characteristics	Purpose/Usage
gem5-Aladdin	Specialized for simulating SoC and accelerators, x86	Hardware-software co-simulation
SMAUG	Optimized for deep learning accelerators, x86	End-to-end acceleration of DL models
gem5-SALAM	Extends gem5 for detailed accelerator modeling, ARM	System-level simulation with accelerators
MAESTRO	Focused on dataflow architecture	Modeling of dataflow accelerators
gem5-X	Advanced extensions to gem5, ARM	General enhancements to gem5 capabilities

- gem5-Aladdin: Known for its efficient simulation of system-on-chip architectures and hardware accelerators. Its ISA is tailored towards accurately modeling the interaction between these components and the rest of the system.
- SMAUG: SMAUG focuses on the simulation of deep learning accelerators, and its ISA is likely optimized for operations commonly found in deep learning models, such as matrix multiplications and convolution operations.
- gem5-SALAM: An extension of gem5, gem5-SALAM introduces additional ISA features for detailed modeling of hardware accelerators. Its ISA probably includes support for a range of accelerator-specific operations, enabling more accurate simulation at the system level.

- MAESTRO: MAESTRO's algorithm is geared towards dataflow architectures, which are commonly used in machine learning accelerators. The algorithm would support operations that are typical in dataflow models, such as parallel data processing and efficient data routing.
- gem5-X: As an advanced version of gem5, gem5-X might include extended ISA capabilities beyond the standard gem5. These enhancements could encompass broader support for various processor architectures and possibly advanced simulation features.

Each framework's chosen ISA reflects its primary purpose, whether it's to simulate specific types of hardware accelerators, to enhance the capabilities of existing simulation environments like gem5, or to focus on particular computational paradigms like dataflow architectures. The selection of an ISA for each framework is made based on criteria unique to the framework, which may be influenced by current trends in architecture research or by the need to compare performance with real-world systems. In some cases, the choice of ISA is also guided by the framework's ability to handle specific features, such as interrupt handling, as seen in frameworks like gem5-SALAM.

Table 4: Supported modes and additional notes for each framework

Framework Name	Syscall Emulation (SE) Mode	Full System (FS) Mode	Notes
gem5-Aladdin	Supported	Unsupported	Primarily for SoC and accelerator simulation
SMAUG	Supported	Might be Supported	Focused on deep learning accelerator simulation
gem5-SALAM	Unsupported	Supported	Extends gem5 for detailed accelerator modeling
MAESTRO	Unknown	Unknown	Specialized in dataflow architecture modeling
gem5-X	Supported	Supported	Advanced extensions of gem5 capabilities

- gem5-Aladdin: This framework, being an extension of gem5, supports only SE mode. Its focus is on simulating interactions between SoCs and accelerators.
- SMAUG: While primarily focused on simulating deep learning accelerators, SMAUG's compatibility with SE and FS modes depends on its integration level with gem5. It's likely compatible with at least SE mode.

- gem5-SALAM: As an extension of gem5, gem5-SALAM supports FS mode. It's designed for detailed simulation and analysis of hardware accelerators within a system.
- MAESTRO: The support for gem5 modes in MAESTRO is less clear, as its primary purpose is to model dataflow architectures, which might not directly align with the typical use cases of SE and FS modes.
- gem5-X: Being an advanced form of gem5, gem5-X supports both SE and FS modes, offering enhanced simulation capabilities and possibly new features beyond the standard gem5 functionalities.

3. RATIONALE FOR SELECTION

3.1 Explanation for choosing gem5-SALAM and gem5-Aladdin for in-depth comparison

Building on the comprehensive comparative analysis conducted in the previous chapters, this chapter focuses on the rationale behind the selection of gem5-Aladdin and gem5-SALAM as the primary frameworks for our subsequent research and experimentation in System-on-Chip design and simulation. These frameworks emerged as the most suitable candidates, each for their distinct strengths and capabilities that align with the specific objectives and criteria of our study.

gem5-Aladdin was selected for its exceptional balance of high simulation accuracy and flexibility. This framework, renowned for its detailed modeling of SoC components, including CPUs and GPUs, offers an ideal platform for conducting precise and in-depth SoC simulations. Its ability to provide accurate performance metrics, power consumption data, and thermal characteristics makes it invaluable for our research, particularly in scenarios requiring detailed co-simulation of CPU-GPU interactions. Furthermore, its user-friendly interface and extensive hardware support significantly enhance the ease of conducting complex simulations, thereby making it a preferred choice for studies that necessitate a comprehensive understanding of SoC architectures.

On the other hand, gem5-SALAM stands out for its very high flexibility and adaptability, coupled with high simulation accuracy. This framework's modular design, featuring components like the CommInterface and LLVMInterface, allows for extensive customization and scalability. It is particularly well-suited for heterogeneous system simulation, offering detailed insights into the operational intricacies of diverse SoC designs. The framework's efficiency in handling various simulation scenarios, from academic research to practical industrial applications, underscores its versatility. Additionally, gem5-SALAM's forward-looking architecture, characterized by its readiness to adapt to future technological advancements, makes it an essential tool for exploratory and innovative research in the field.

The decision to select gem5-Aladdin and gem5-SALAM frameworks was based on their strengths and alignment with our research goals. Our comparative study involves evaluating these frameworks through experiments that focus on performance metrics, power consumption, and broader SoC architectures. gem5-Aladdin excels in detailed architectural analysis and CPU-GPU co-simulation, while gem5-SALAM's flexibility caters to innovative SoC designs. Integrating LLVM enhances their functionality, streamlining the simulation process in gem5-Aladdin and providing adaptability in gem5-SALAM. Our research aims to provide a comprehensive understanding of each framework's capabilities and limitations, guiding informed decisions in SoC simulation and contributing to advancements in both academic and industrial settings.

3.2 In-Depth Power Consumption Study

The significance of energy efficiency in SoC design cannot be overstated. Building upon our initial power consumption observations, the thesis aims to delve into a comprehensive study of how each framework's power efficiency scales with different workloads and configurations. This series of experiments will not only measure static power consumption but also examine dynamic power management under diverse operational loads. Such an analysis will offer a holistic view of the power management strategies employed by gem5-Aladdin and gem5-SALAM, informing their suitability for energy-sensitive applications.

3.3 Scalability and Adaptability Exploration

Furthering our investigation into the area requirements of each framework, the thesis plans to conduct comparative studies focusing on scalability and adaptability. As SoC designs grow in complexity, understanding how a simulation framework manages space becomes increasingly crucial. The thesis will analyze area utilization across various SoC architectures, examining how each framework responds to changes in design specifications. These studies are critical in assessing the scalability and adaptability of gem5-Aladdin and gem5-SALAM, key factors in their application to diverse design scenarios.

3.4 Integration and Usability Assessment

To round out our comparative analysis, the thesis will explore the integration capabilities and user experience of both frameworks. This involves evaluating their interoperability with other standard tools and systems in SoC design and evaluating their accessibility and usability. Understanding these aspects is essential in determining the overall user experience and readiness for adoption in both academic and industry settings.

3.5 Looking Ahead: Anticipating Challenges and Outcomes

As the thesis examines deeper these advanced experiments, it anticipates encountering new challenges, particularly regarding the complexity of configurations and potential variability in results. The thesis will address these challenges through meticulous planning, documentation, and multiple iterations of experiments to ensure consistency and reliability.

3.6 Comparison and Experimental Measurement Evaluation

- **Accelerator Clock Cycles:** The detailed comparison takes a close look at operational efficiency, particularly concerning accelerator clock cycles. By evaluating how each framework counts clock cycles under different workloads and settings, this study aims to provide a comprehensive insight into their timing efficiency.

- **Power Consumption:** The in-depth power consumption study suggested is specifically designed to build upon the initial power observations. By examining how

each framework manages power under various operational loads and configurations, this study aims to provide a comprehensive understanding of their energy efficiency.

- **Area Requirements:** The scalability and adaptability exploration addresses the frameworks' management of space, a critical factor in SoC design. This involves analyzing area utilization across different SoC architectures, which aligns with the area requirements data from your experiments.

- **Total Clock Cycles and Instructions:** These metrics are part of the overall performance analysis. The suggested experiments aim to provide deeper insights into the total clock cycles and instructions, which are integral to assessing the overall efficiency and performance of the frameworks.

- **Total Simulation Time:** The advanced comparative experiments are designed to also consider the total simulation time. This involves understanding how each framework performs under different simulation durations and conditions, which is important for evaluating their practical applicability in real-world scenarios.

- **Comparison between CPU only and CPU with Accelerator:** The suggested experiments include an assessment of how well each framework integrates with other systems, as well as their adaptability to various SoC architectures. This directly relates to comparing CPU-only versus CPU and accelerator configurations, providing insights into each framework's performance in different hardware setups.

4. EXPERIMENTAL SETUP & EVALUATION

4.1 Experimental Setup

The section presents the setup and configuration processes for the gem5-SALAM and gem5-Aladdin frameworks.

4.1.1 Installation Procedure

The foundational step in this process involves the installation of Docker, a pivotal platform that simplifies the deployment, execution, and management of applications within containerized environments. The SMAUG framework is accessible as a Docker image, which ensures a standardized and consistent setup procedure.

Docker Installation

The initial phase entails installing Docker on the system, a prerequisite for the execution of the SMAUG framework. Docker can be acquired and installed from its official website, following the guidelines provided therein.

Retrieving the SMAUG Docker Image

After the installation of Docker, the next step is to retrieve the latest SMAUG Docker image using the command:

```
docker pull xyzsam/smaug:latest
```

Initiating the Docker Container

Post retrieval of the image, the Docker container is initiated using the command below:

```
docker run -it --rm --mount source=smaug-workspace,target=/workspace xyzsam/smaug:latest
```

This command establishes a Docker volume (smaug-workspace) that is essential for retaining all source codes, modifications, and build artifacts securely.

4.1.2 Source Code Repository Updates

Within the Docker container, it is imperative to navigate to the workspace and update the necessary repositories with the following set of commands:

```
cd gem5-aladdin && git pull origin master && git submodule update --init --recursive && cd ..
```

```
cd LLVM-Tracer && git pull origin master && cd ..
```

```
cd smaug && git pull origin master && git submodule update --init --recursive && cd ..
```

These commands ensure the acquisition of the most recent versions of the gem5-Aladdin, LLVM-Tracer, and SMAUG repositories, which are crucial for the framework's functionality.

4.1.3 Constructing the Simulation Environment Building gem5-Aladdin

The next phase involves navigating to the gem5-Aladdin directory and constructing it using the command delineated below:

```
cd /workspace/gem5-aladdin
```

```
python2.7 `which scons` build/X86/gem5.opt PROTOCOL=MESI_Two_Level_aladdin -j2
```

The -j parameter in this context denotes the number of CPUs allocated for the build process and can be adjusted in accordance with the capabilities of the underlying system.

Compiling SMAUG

Following the construction of gem5-Aladdin, the subsequent step is to compile SMAUG with the command:

```
cd /workspace/smaug
```

```
make all -j8
```

This command instigates the compilation of all the essential components of the SMAUG framework.

4.1.4 Executing the Initial Model

To validate the installation and setup, it is recommended to execute a pre-defined model, such as the 4-layer Minerva model with the NVDLA-like backend, codenamed SMV:

```
cd /workspace/smaug/experiments/sims/smv/tests/minerva
```

```
sh run.sh
```

The execution of this script commences the simulation of the Minerva model, serving as a practical test to ascertain the correctness and efficacy of the setup.

4.1.5 Adaptation for Alternative Benchmarks

Adapting the setup for other benchmarks, such as gemm, necessitates the modification of configuration files accordingly. This involves the creation of a new configuration file (test_gemm.cfg), the update of the gem5.cfg file to reference this new configuration, and the generation of the dynamic trace and instrumented binary for gemm. The Xenon system can facilitate much of this process through automation:

```
cd /workspace/gem5-aladdin/sweeps
```

```
python generate_design_sweeps.py benchmarks/machsuite.xe
```

Executing this command in the sweeps directory generates the required configuration files and traces for the benchmark.

The process of setting up the environment for gem5-Aladdin is characterized by a series of crucial steps, each contributing to a fully functional simulation platform. Initially, the procedure commences with the installation of Docker, a foundational tool for containerizing applications. Following this, the SMAUG Docker image is obtained and activated, setting the stage for the subsequent phases. The next step involves updating various source code repositories, ensuring that the latest versions of gem5-Aladdin, LLVM-Tracer, and SMAUG are in place. Subsequently, the focus shifts to building the gem5-Aladdin and SMAUG frameworks, a critical phase where the simulation tools are compiled and prepared for use. Finally, the environment's readiness is validated by running a predefined test model, typically the Minerva model with the NVDLA-like backend SMV. This orchestrated sequence of steps culminates in a comprehensive environment, ready for researchers to simulate deep learning models on diverse SoC configurations, thereby facilitating intricate hardware-software co-design tasks within deep neural network research. The terminal output from the execution of a simulation using the gem5-Aladdin framework:

```
================================
        Aladdin Results
================================
Running : /workspace/gem5-aladdin/sweeps/machsuite/fft_strided/0/outputs/fft_strided
Top level function: fft
Cycle : 108596 cycles
Upsampled Cycle : 0 cycles
Avg Power: 51.319 mW
Idle FU Cycles: 39963 cycles
Avg FU Power: 46.8043 mW
Avg FU Dynamic Power: 42.7155 mW
Avg FU leakage Power: 4.0888 mW
Avg MEM Power: 4.51466 mW
Avg MEM Dynamic Power: 3.1622 mW
Avg MEM Leakage Power: 1.35246 mW
Total Area: 410559 uM^2
FU Area: 326932 uM^2
MEM Area: 83627 uM^2
Num of Double Precision FP Multipliers: 4
Num of Double Precision FP Adders: 4
Num of Bit-wise Operators (32-bit): 3
Num of Shifters (32-bit): 1
Num of Registers (32-bit): 1034
================================
        Aladdin Results
================================
Success.
Exiting @ tick 15933045128 because exiting with last active thread context
```

Figure 7: Screenshots of gem5-Aladdin indicative benchmark results.

```
---------- Begin Simulation Statistics ----------
# Stats desc: fft_strided_datapath completed.
final_tick                                 5120490000                       # Number of ticks from beginning of simulation (restored from checkpoints and never rese
host_inst_rate                               68056                          # Simulator instruction rate (inst/s)
host_mem_usage                             4709688                          # Number of bytes of host memory used
host_op_rate                                147824                          # Simulator op (including micro ops) rate (op/s)
host_seconds                                 67.45                          # Real time elapsed on the host
host_tick_rate                             75910008                         # Simulator tick rate (ticks/s)
sim_freq                                1000000000000                       # Frequency of simulated ticks
sim_insts                                  4590724                          # Number of instructions simulated
sim_ops                                    9971442                          # Number of ops (including micro ops) simulated
sim_seconds                                0.005120                         # Number of seconds simulated
sim_ticks                                  5120490000                       # Number of ticks simulated
system.cpu.branchPred.BTBCorrect                 0                          # Number of correct BTB predictions (this stat may not work properly.
system.cpu.branchPred.BTBHitPct            0.000000                         # BTB Hit Percentage
system.cpu.branchPred.BTBHits                    0                          # Number of BTB hits
system.cpu.branchPred.BTBLookups              1363501                         # Number of BTB lookups
system.cpu.branchPred.RASInCorrect               1                          # Number of incorrect RAS predictions.
system.cpu.branchPred.condIncorrect             58062                        # Number of conditional branches incorrect
system.cpu.branchPred.condPredicted           1731216                        # Number of conditional branches predicted
system.cpu.branchPred.indirectHits             992373                        # Number of indirect target hits.
system.cpu.branchPred.indirectLookups         1363501                        # Number of indirect predictor lookups.
system.cpu.branchPred.indirectMisses           371128                        # Number of indirect misses.
system.cpu.branchPred.lookups                 1836635                        # Number of BP lookups
system.cpu.branchPred.usedRAS                   49212                        # Number of times the RAS was used to get a target.
system.cpu.branchPredindirectMispredicted       34051                        # Number of mispredicted indirect branches.
system.cpu.cc_regfile_reads                   7816049                        # number of cc regfile reads
system.cpu.cc_regfile_writes                  3536970                        # number of cc regfile writes
system.cpu.commit.amos                           0                           # Number of atomic instructions committed
system.cpu.commit.branchMispredicts             58086                        # The number of times a branch was mispredicted
system.cpu.commit.branches                    1400975                        # Number of branches committed
system.cpu.commit.bw_lim_events                463505                        # number cycles where commit BW limit reached
```

Figure 8: Screenshots of gem5-Aladdin indicative benchmark results

Figure 7 & 8 provide a detailed summary of the simulation's performance in terms of execution cycles, power consumption, area utilization, and computational resource usage. Additionally, they offer insights into the accuracy of the simulated CPU's branch prediction mechanism and the general operation of the simulation. This data is critical for evaluating the performance and efficiency of the hardware being simulated and for making decisions regarding potential optimizations in hardware design.

In Figure 7, the Aladdin Results section outlines various performance metrics:

- Cycle: The total number of cycles taken to complete the FFT operation is listed as 108596 cycles.
- Average Power Consumption: The average power consumed by the functional units (FUs) is 46.8043 mW, with the dynamic and leakage power detailed separately.
- Memory Power Consumption: The memory power usage is also reported, with average, dynamic, and leakage power consumption values.
- Total Area: The total area used by the FUs and memory is reported in micrometers squared (μm²).
- Functional Unit and Memory Areas: The specific areas for functional units and memory are detailed separately.
- Count of Double Precision Multipliers and Adders: The simulation used 4 double precision floating-point (FP) multipliers and adders.
- Count of Bit-wise Operators: The number of bit-wise operators, shifters, and registers are also reported, indicating the resources allocated for computation.

Figure 8 contains the Simulation Statistics from gem5, which includes:

- Simulation Time and Rate: The total number of simulated ticks, instruction rate, and operation rate are listed, providing an insight into the simulation's execution speed.
- Branch Prediction Statistics: A series of statistics related to the branch prediction mechanism of the simulated CPU, such as the number of correct/incorrect predictions, hit/miss rates, and lookups.
- Instruction and Operation Counts: The total number of instructions and operations (including micro-operations) simulated are presented.
- Commit Events: Various events related to the commit stage in the CPU pipeline are listed, such as the number of branches committed and branch mispredictions.

4.1.6 Commencement of Installation Protocols

The initiation of the installation procedure for the gem5-SALAM environment is marked by the methodical installation of prerequisite software packages on the Ubuntu 20.04 LTS operating system. This collection, comprising compilers, libraries, and essential tools, constitutes the foundational elements necessary for the successful compilation and subsequent execution of the gem5-SALAM framework. The execution of the installation commands is prescribed as follows:

```
sudo apt install build-essential git m4 scons zlib1g zlib1g-dev \
```

```
libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev \
python3-dev python-is-python3 libboost-all-dev pkg-config
```

4.1.7 Configuration and Deployment of LLVM/Clang

LLVM and Clang constitute the core elements in the gem5-SALAM environment, orchestrating the compilation processes and the generation of LLVM Intermediate Representation (IR). The deployment of LLVM-12 and Clang-12 can be effectuated through the ensuing command:

```
sudo apt install llvm-12 llvm-12-tools clang-12
```

Subsequent to the installation, it is incumbent upon the user to execute the script for updating alternatives, as delineated in the gem5-SALAM documentation, to ensure the system's proper configuration.

4.1.8 Repository Acquisition and Framework Compilation

With the preliminary dependencies resolved, the user shall proceed to clone the gem5-SALAM repository from its repository on GitHub:

```
git clone https://github.com/TeCSAR-UNCC/gem5-SALAM
```

the compilation of the simulation binaries is a pivotal phase. Utilizing scons, the binaries are constructed with an emphasis on the optimization for parallel compilation. The 'opt' and 'debug' builds are the recommended types for development:

```
scons build/ARM/gem5.opt -j$(nproc)
```

```
scons build/ARM/gem5.debug -j$(nproc)
```

4.1.9 Execution of gem5-SALAM

Upon the completion of the build process, the user must delineate the computational model of the accelerator. This is achieved by scripting the desired functionalities in a preferred language, which is then translated into LLVM IR. The LLVM IR is integrated into an LLVMInterface within gem5-SALAM, followed by the association of the accelerator with the appropriate communication interface within the memory map.

For the purposes of system validation, examples are accessible within the benchmarks/sys_validation directory. To compile these benchmarks, the M5_PATH environment variable must be set, and the compilation is carried out thusly:

```
export M5_PATH=/path/to/gem5-SALAM
```

```
cd $M5_PATH/benchmarks/sys_validation/[benchmark]
```

```
make
```

To run the benchmarks, one invokes the *run_system.sh* script with the appropriate parameters:

```
$M5_PATH/tools/run_system.sh --bench [benchmark name] --bench-path [path]
```

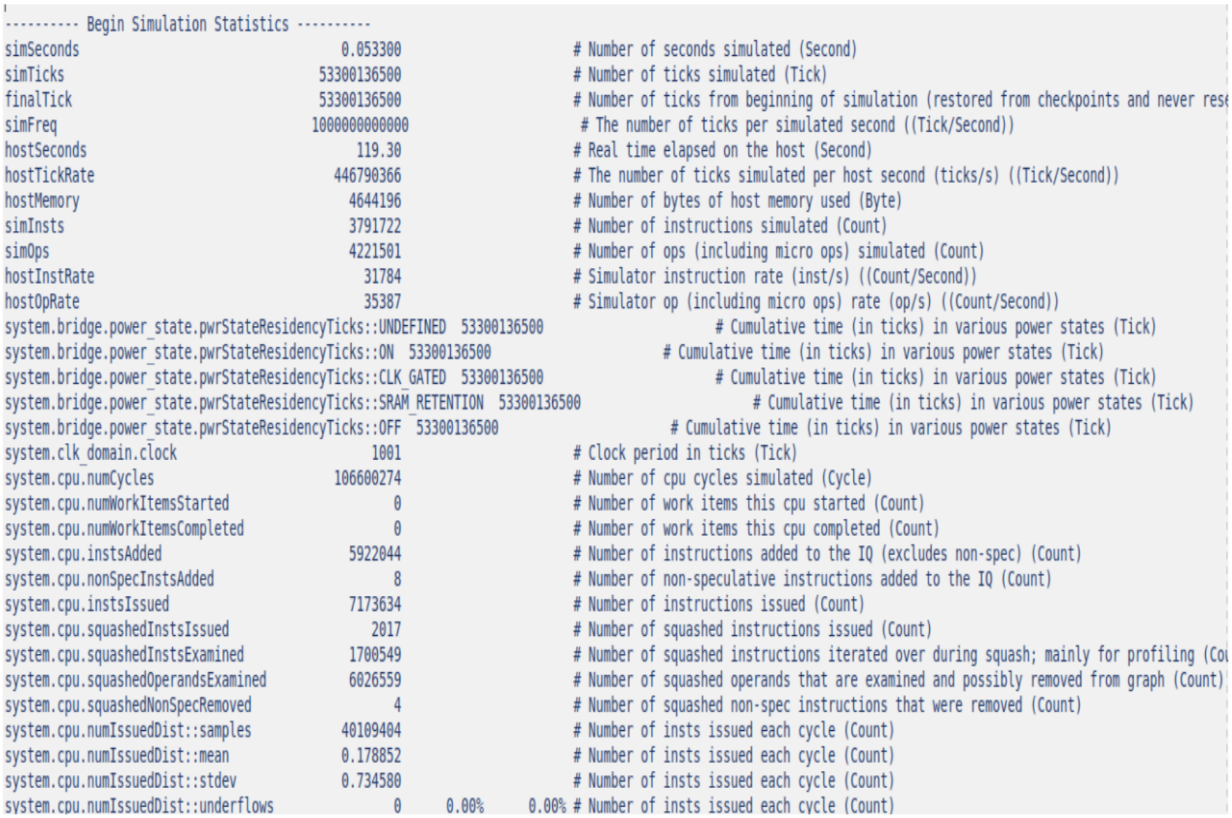
4.1.10 Modification of Benchmarks and Analysis of Results

For CPU-only benchmark execution, alterations to the original code are necessitated. This process entails the deactivation of the commands initiating the accelerator and their substitution with the primary computational logic.

The fruits of the simulation—detailed performance metrics—are harvested from the standard output and the stats.txt file generated by gem5.

To encapsulate, the configuration of the environment for gem5-SALAM and gem5-Aladdin is a multi-faceted and structured undertaking that involves the installation of essential software components, the assembly of the simulation framework, and the diligent execution of benchmarks. Adherence to the procedures specified within this chapter is critical for the successful instantiation of a simulation environment, pivotal for the exploration and assessment of SoC designs and the efficacious simulation of deep learning models on bespoke hardware accelerators.

Here follows the images of indicative of a benchmarking output from a simulation environment gem5-SALAM:



```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.053300                       # Number of seconds simulated (Second)
simTicks                                  53300136500                       # Number of ticks simulated (Tick)
finalTick                                 53300136500                       # Number of ticks from beginning of simulation (restored from checkpoints and never rese
simFreq                                1000000000000                        # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                  119.30                       # Real time elapsed on the host (Second)
hostTickRate                              446790366                       # The number of ticks simulated per host second (ticks/s) ((Tick/Second))
hostMemory                                  4644196                       # Number of bytes of host memory used (Byte)
simInsts                                    3791722                       # Number of instructions simulated (Count)
simOps                                      4221501                       # Number of ops (including micro ops) simulated (Count)
hostInstRate                                  31784                       # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                    35387                       # Simulator op (including micro ops) rate (op/s) ((Count/Second))
system.bridge.power_state.pwrStateResidencyTicks::UNDEFINED  53300136500                  # Cumulative time (in ticks) in various power states (Tick)
system.bridge.power_state.pwrStateResidencyTicks::ON  53300136500                  # Cumulative time (in ticks) in various power states (Tick)
system.bridge.power_state.pwrStateResidencyTicks::CLK_GATED  53300136500                  # Cumulative time (in ticks) in various power states (Tick)
system.bridge.power_state.pwrStateResidencyTicks::SRAM_RETENTION  53300136500                  # Cumulative time (in ticks) in various power states (Tick)
system.bridge.power_state.pwrStateResidencyTicks::OFF  53300136500                  # Cumulative time (in ticks) in various power states (Tick)
system.clk_domain.clock                        1001                       # Clock period in ticks (Tick)
system.cpu.numCycles                        106600274                       # Number of cpu cycles simulated (Cycle)
system.cpu.numWorkItemsStarted                    0                       # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted                  0                       # Number of work items this cpu completed (Count)
system.cpu.instsAdded                       5922044                       # Number of instructions added to the IQ (excludes non-spec) (Count)
system.cpu.nonSpecInstsAdded                      8                       # Number of non-speculative instructions added to the IQ (Count)
system.cpu.instsIssued                      7173634                       # Number of instructions issued (Count)
system.cpu.squashedInstsIssued                 2017                       # Number of squashed instructions issued (Count)
system.cpu.squashedInstsExamined            1700549                       # Number of squashed instructions iterated over during squash; mainly for profiling (Cou
system.cpu.squashedOperandsExamined         6026559                       # Number of squashed operands that are examined and possibly removed from graph (Count)
system.cpu.squashedNonSpecRemoved                 4                       # Number of squashed non-spec instructions that were removed (Count)
system.cpu.numIssuedDist::samples           40109404                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::mean              0.178852                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::stdev             0.734580                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::underflows              0      0.00%      0.00% # Number of insts issued each cycle (Count)
```

Figure 9: Screenshots of gem5-SALAM's indicative benchmark results

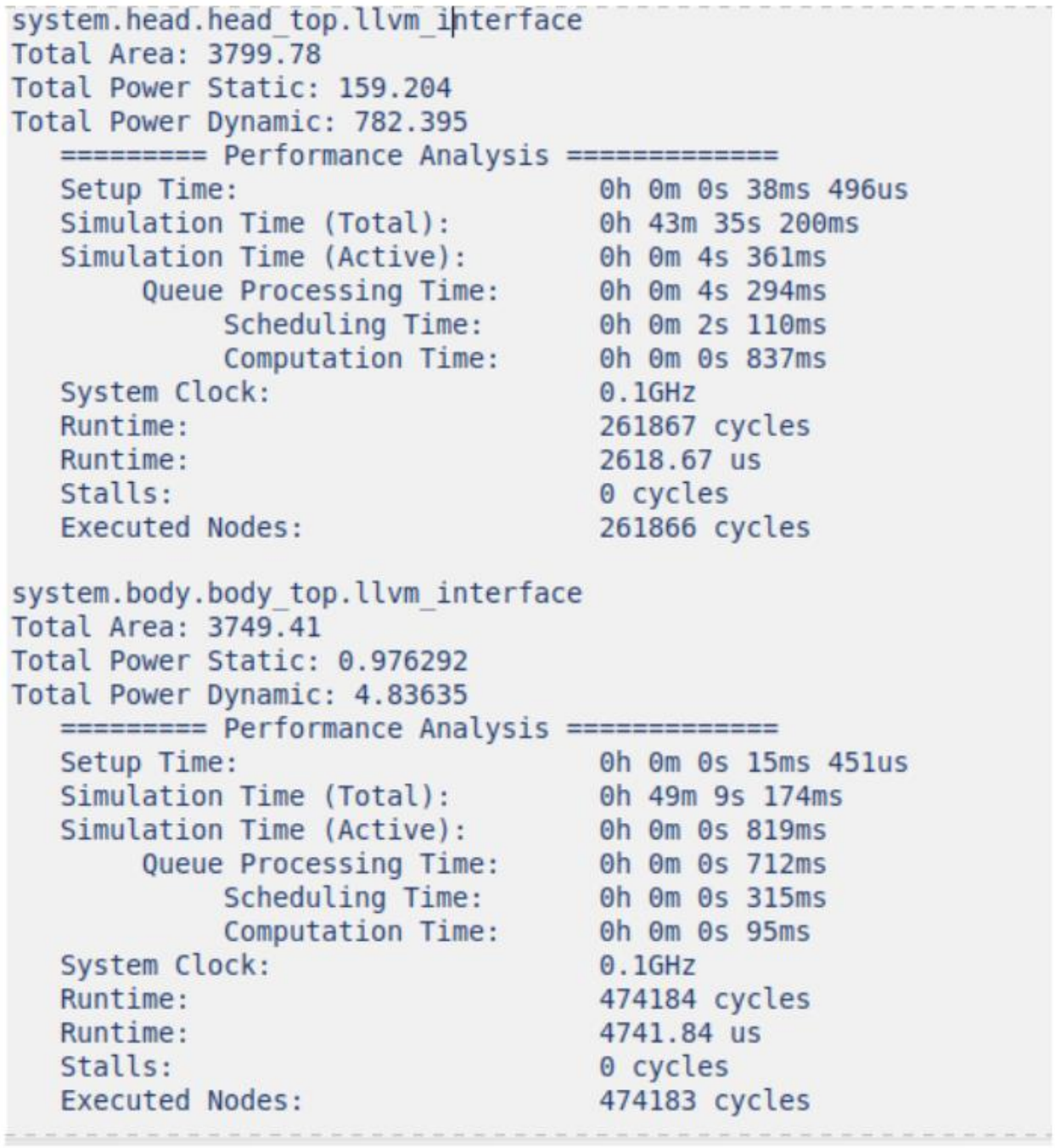


Figure 10: Screenshots of gem5-SALAM's indicative benchmark results.

In Figure 9, the simulation statistics begin with a header "---------- Begin Simulation Statistics ----------" and provide a detailed account of the simulation's execution. Key metrics include:

- simSeconds: The total simulated time in seconds.
- simTicks: The total number of ticks (the basic time unit in the simulation) that have been simulated.
- finalTick: The final tick count at the end of simulation.
- simFreq: The frequency of simulated ticks.
- hostSeconds: The real-world time that has elapsed during the simulation.
- hostTickRate: The number of ticks simulated per second of real-world time.
- hostMemory: The amount of host memory used by the simulation.
- simInsts: The total number of instructions simulated.
- simOps: The total number of operations (including micro-operations) simulated.
- hostInstRate and hostOpRate: The rates at which instructions and operations are simulated.

It also includes metrics related to the power state and residency ticks of various components, like the system bridge, and core CPU statistics like the number of cycles, work items started and completed, and instruction details. Additionally, there are CPU stats like numInstsAdded, instsIssued, and squashedInstsIssued which are likely related to the CPU's pipeline operation, including how many instructions are added to issue queues, how many are actually issued, and how many are squashed and not executed, respectively.

The second screenshot focuses more on the performance analysis of specific components, which suggests that they might be the simulated hardware accelerators or parts of the system being tested. Metrics such as total area, total static, and dynamic power provide a direct measure of the physical and power consumption characteristics of these components.

Furthermore, the performance analysis section breaks down the simulation and setup times, indicating the efficiency of the simulation and providing data that can be used to optimize the simulation process or the hardware being simulated.

These benchmarks are critical for gem5-SALAM, as they allow for the evaluation of the simulated accelerators' performance against various metrics such as power consumption, area, and processing time. Such information is invaluable for architects and system designers who aim to optimize their hardware designs based on simulation feedback.

4.2 Details of the evaluation process

In this chapter, the thesis examines a systematic process to test the simulation environment established in the previous chapter. The focus of our examination is two-fold: to validate the functional correctness of the system under simulation and to benchmark its performance under various operational conditions.

4.2.1 Evaluation Strategy

Our evaluation strategy was predicated on a rigorous, repeatable approach that would allow for an in-depth analysis of the system's performance. The thesis employed a suite of benchmarks, carefully chosen for their relevance to the system's intended application domain, ensuring a comprehensive evaluation across a broad spectrum of scenarios.

4.2.2 Benchmark Selection and Justification

The benchmarks were selected based on specific criteria: their ability to effectively stress test the system's computational capabilities, their common use in the industry as standard performance indicators, and their coverage of the architectural features pertinent to our research. Each benchmark was chosen to represent a unique aspect of system performance, such as computational throughput, memory access patterns, or power efficiency.

4.2.3 Test Environment and Configuration

The thesis meticulously configured the test environment to mirror the conditions under which the system would operate in a real-world scenario. This included setting up the gem5-SALAM and gem5-Aladdin simulation parameters to reflect realistic hardware constraints and workload characteristics.

4.2.4 Execution of Benchmarks

The benchmarks were executed systematically, with each run meticulously documented to capture all relevant performance data. The thesis ensured the consistency of the evaluation conditions to allow for accurate comparisons across different simulation runs.

4.2.5 Data Collection Methodology

Data was collected via the gem5 standard output logs and the stats.txt file, which provided a granular view of the system's operational characteristics. The thesis parsed and aggregated this data, facilitating a structured approach to data analysis.

4.2.6 Analysis of Test Results

The thesis employed statistical analysis tools to interpret the collected data, allowing us to draw meaningful conclusions about the system's performance. This included the use of average, median, and standard deviation measures to understand the distribution and variability of our results.

4.2.7 Addressing Anomalies and Inconsistencies

Throughout the evaluation process, the thesis encountered and addressed various anomalies and inconsistencies. These were analyzed to understand their root causes, and

appropriate adjustments were made to the evaluation protocol to mitigate their impact on the results.

4.2.8 Implications of Findings

The findings from our evaluation process provided valuable insights into the system's performance and potential areas for optimization. These insights have implications for future design iterations and for the broader field of SoC and accelerator design.

4.3 Modifications Made to Run Common Benchmarks for Analysis and Comparison

Modifications to common benchmarks (MachSuite benchmarks [36]) for analysis and comparison in simulation environments like gem5-SALAM and gem5-Aladdin are made to ensure that the benchmarks accurately reflect the design and operational nuances of the systems under study. These adjustments are necessary to validate the functionality and performance of various hardware accelerator designs.

In the context of benchmark modifications for the gem5-SALAM environment, additional adjustments are often required due to the absence of ready-made code for CPU-only runs, which is a notable difference from Aladdin. In scenarios where only CPU performance needs to be gauged—without the influence of an accelerator—modifications include the deactivation of accelerator calls within the benchmark code. This involves commenting out or removing the specific commands that invoke the accelerator and inserting equivalent CPU-based processing code segments to replace the accelerator's functionality.

Moreover, due to gem5-SALAM's dependencies on "pinned" memory regions, benchmarks may require alterations to manage dynamic memory effectively. This ensures that the simulations accurately reflect the performance of the CPU alone, without any enhancements that would typically be contributed by the hardware accelerators. These changes are crucial for achieving a clear comparative analysis between CPU-only and accelerator-enhanced performances, providing a baseline for measuring the impact of the accelerator's integration into the system.

Through this meticulous process of adapting benchmarks, this thesis can conduct a granular analysis of the system's performance, understanding the distinct contributions of CPUs and accelerators to the overall computational workload. These insights form the cornerstone for optimization efforts, enabling the refinement of both the simulated hardware accelerators and the system architecture.

Here are some images of the latest modification:

```
    double* real        = (double *)malloc(FFT_SIZE*sizeof(double));
    double *img         = (double *)malloc(IMG_OFFSET*sizeof(double));
    double *real_twid   = (double *)malloc(RTWID_OFFSET*sizeof(double));
    double *img_twid    = (double *)malloc(ITWID_OFFSET*sizeof(double));
    double *real_check  = (double *)malloc(RCHK_OFFSET*sizeof(double));
    double *img_check   = (double *)malloc(ICHK_OFFSET*sizeof(double));

    volatile int count = 0;
    stage = 0;

    ffts.real       = real;
    ffts.img        = img;
    ffts.real_twid  = real_twid;
    ffts.img_twid   = img_twid;
    ffts.real_check = real_check;
    ffts.img_check  = img_check;

    printf("Generating data\n");
    genData(&ffts);
    printf("Data generated\n");
/*
    *loc_real       = (uint32_t)(void *)real;
    *loc_img        = (uint32_t)(void *)img;
    *loc_real_twid  = (uint32_t)(void *)real_twid;
    *loc_img_twid   = (uint32_t)(void *)img_twid;

    *top = 0x01;
    while (stage < 1) count++;

    printf("Job complete\n");
*/
```

Figure 11: Code change to run benchmark without accelerator.

```
    printf("CPU process started\n");

    ffts.real       = real;
    ffts.img        = img;
    ffts.real_twid  = real_twid;
    ffts.img_twid   = img_twid;
    ffts.real_check = real_check;
    ffts.img_check  = img_check;

    printf("Generating data\n");
    genData(&ffts);
    printf("Data generated\n");

    int even, odd, span, log, rootindex;
    double temp;

    log = 0;

    outer:
    for(span=FFT_SIZE>>1; span; span>>=1, log++){
        inner:
        for(odd=span; odd<FFT_SIZE; odd++){
            odd |= span;
            even = odd ^ span;

            temp = real[even] + real[odd];
            real[odd] = real[even] - real[odd];
            real[even] = temp;

            temp = img[even] + img[odd];
            img[odd] = img[even] - img[odd];
            img[even] = temp;

            rootindex = (even<<log) & (FFT_SIZE - 1);
            if(rootindex){
```

Figure 12: Code change to run benchmark without accelerator.

4.4 Any challenges or collisions encountered during the experiments

In the pursuit of assessing the system's robustness via benchmark compilation and execution, one may encounter a spectrum of compatibility issues or unexpected runtime errors. These obstacles are not uncommon in complex simulation environments where myriad factors, such as system configurations, library dependencies, and hardware specifications, interplay.

However, as part of a diligent evaluation regimen, such issues were promptly addressed. This involved a systematic approach to problem-solving which included examining compiler warnings and errors, scrutinizing log files for runtime exceptions, and validating environment settings. Each problem was dissected to understand its root cause—be it a mismatch in

expected data formats, an oversight in memory management, or a subtle bug within the codebase.

The resolution process was often as instructional as the benchmarking itself. It necessitated a deep dive into the intricacies of the simulation framework and the hardware models being emulated. By iteratively refining the code and configuration, a stable execution pathway was forged, ensuring that the benchmarks ran smoothly.

This process underscored the importance of resilience and adaptability in research. While the details of these compatibility and runtime challenges were not meticulously chronicled due to their swift resolution, the enhanced understanding gained from addressing these issues was invaluable. It not only facilitated the immediate progress of the benchmarks but also bolstered the simulation environment's reliability for future computational explorations.

5. RESULTS AND REVIEW (ANALYSIS)

5.1 Types of benchmarks used for analysis

The analysis in question utilizes a diverse array of benchmarks to evaluate the performance of computing systems. These benchmarks—bfs (Breadth-First Search), fft (Fast Fourier Transform), gemm (General Matrix Multiply), md_knn (Molecular Dynamics K-Nearest Neighbors), mergesort, nw (Needleman-Wunsch), spmv (Sparse Matrix-Vector Multiplication), stencil2d, and stencil3d—are specifically chosen to represent a wide range of computational tasks, each with unique characteristics and demands. This comprehensive selection ensures a thorough assessment of the system's capabilities across various computational scenarios. The benchmarks were run using the gem5-SALAM and SMAUG tools, with SMAUG being used because it includes the latest and most robust version of gem5-Aladdin, providing a reliable and accurate simulation environment for these performance evaluations.

- **Breadth-First Search (bfs):** BFS is a fundamental algorithm in graph theory used to traverse or search tree or graph data structures. It's a cornerstone in numerous computational tasks, especially in areas like networking, pathfinding algorithms in games, and social networking services. Evaluating BFS performance is critical in understanding how well a system can handle data structures and algorithms fundamental to graph theory.
- **Fast Fourier Transform (fft):** FFT is an algorithm to compute the Discrete Fourier Transform (DFT) and its inverse. FFT is widely used in engineering, science, and mathematics. It's crucial in signal processing, image analysis, and solving partial differential equations. The efficiency of FFT computation is a key metric for systems used in digital signal processing and related fields.
- **General Matrix Multiply (gemm):** GEMM forms the core of many scientific and engineering simulations. It's a building block for linear algebra operations, deep learning computations, and much more. The ability to efficiently perform matrix multiplication is vital for systems designed for high-performance computing and artificial intelligence applications.
- **Molecular Dynamics K-Nearest Neighbors (md_knn):** This benchmark is pertinent in the field of molecular dynamics, where simulations of the physical movements of atoms and molecules are crucial. K-Nearest Neighbors (KNN) is a simple, yet effective algorithm used in pattern recognition and classification. Evaluating md_knn performance provides insights into how well a system can handle complex simulations and data-driven algorithms.
- **Mergesort:** As a classic sorting algorithm, mergesort is critical in understanding a system’s efficiency in sorting operations. It's a fundamental algorithm used in database management and data analysis tasks. The performance of mergesort is a good indicator of how a system handles divide-and-conquer strategies and data manipulation tasks.
- **Needleman-Wunsch (nw):** NW is a widely used algorithm in bioinformatics, specifically for sequence alignment. It’s crucial in genetic research and understanding

biological processes. The benchmark evaluates how well a system can perform in the computational biology domain, where sequence alignment is a common task.

- **Sparse Matrix-Vector Multiplication (spmv):** SPMV is a key operation in many scientific and engineering applications, especially where large sparse matrices are involved. It's critical in simulations, optimizations, and solving systems of linear equations. SPMV performance is a vital metric for systems used in scientific computing.
- **Stencil2d and Stencil3d:** These benchmarks involve operations on multi-dimensional data grids and are representative of computations in areas like image processing, computational fluid dynamics, and physical simulations. They test a system's ability to handle complex data structures and perform repeated operations over these structures.

In summary, the selection of these benchmarks is strategic and deliberate. Each benchmark serves to test different aspects of system performance, from data processing and manipulation to algorithmic efficiency in various domains. This broad spectrum of tests ensures a well-rounded evaluation, providing valuable insights into the strengths and weaknesses of the computing systems under study. By analyzing performance across these benchmarks, researchers can identify areas for optimization, leading to more efficient and effective computing solutions. This analysis is not just a testament to the systems' current capabilities but also a guidepost for future improvements and innovations.

5.2 Presentation of results through diagrams and charts

i) Statistics results for accelerator only

Figure 13 shows the clock cycles of 9 accelerator designs between gem5-Aladdin and gem5-SALAM. We can see that there is an outlier for the GEMM accelerator design when it runs on gem5-Aladdin. In this design, the number of cycles is more than 3x higher than when executed in gem5-SALAM. This discrepancy can be attributed to the fact that GEMM is a particularly demanding benchmark, placing significant overhead on caches. In gem5-SALAM, the management of these resources can be performed more efficiently, leading to improved performance. For every other accelerator design, we observe that the number of clock cycles is very close between these two frameworks.

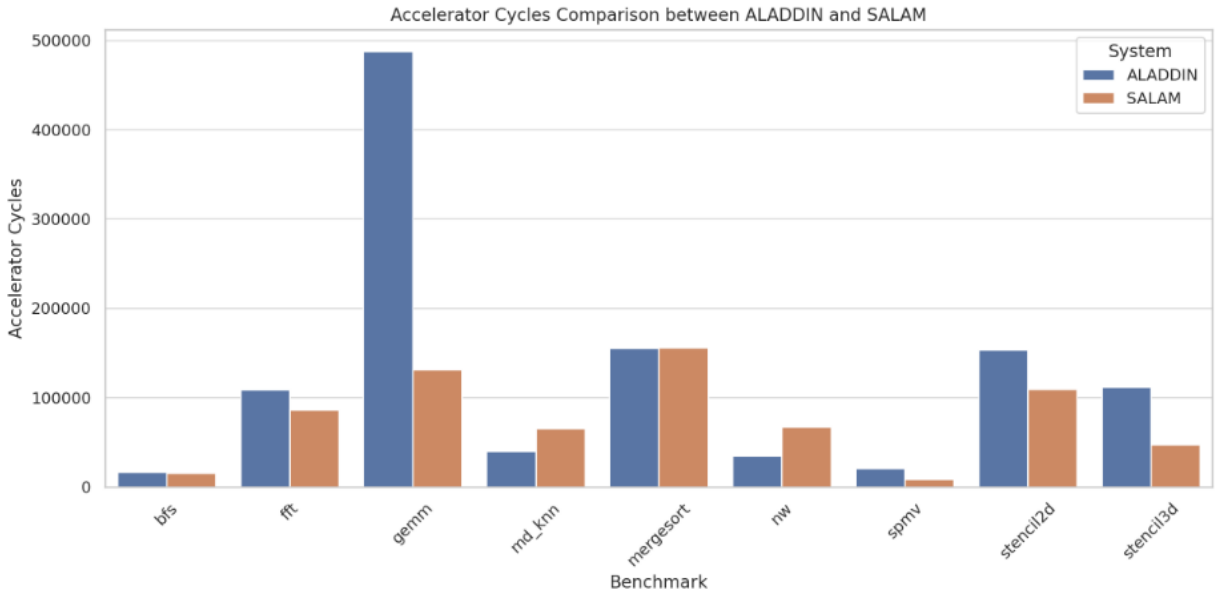


Figure 13: Accelerator cycles comparison between gem5-Aladdin and gem5-SALAM.

Figure 14 & 15 present a power usage data for gem5-Aladdin and gem5-SALAM respectively across various benchmarks. It is important to clarify that these frameworks are based on fundamentally different systems and architectures. Therefore, this is not a direct comparison, but rather an illustration of the magnitude and patterns of power consumption for each framework individually. Essentially, it highlights a pronounced difference in each benchmark, where the power consumption in the gem5-SALAM framework is higher compared to gem5-Aladdin, indicating that the power consumption computation in gem5-SALAM may not be very accurate. As mentioned later, these frameworks also employ different methodologies for calculating power, which further underscores that they are not directly comparable.

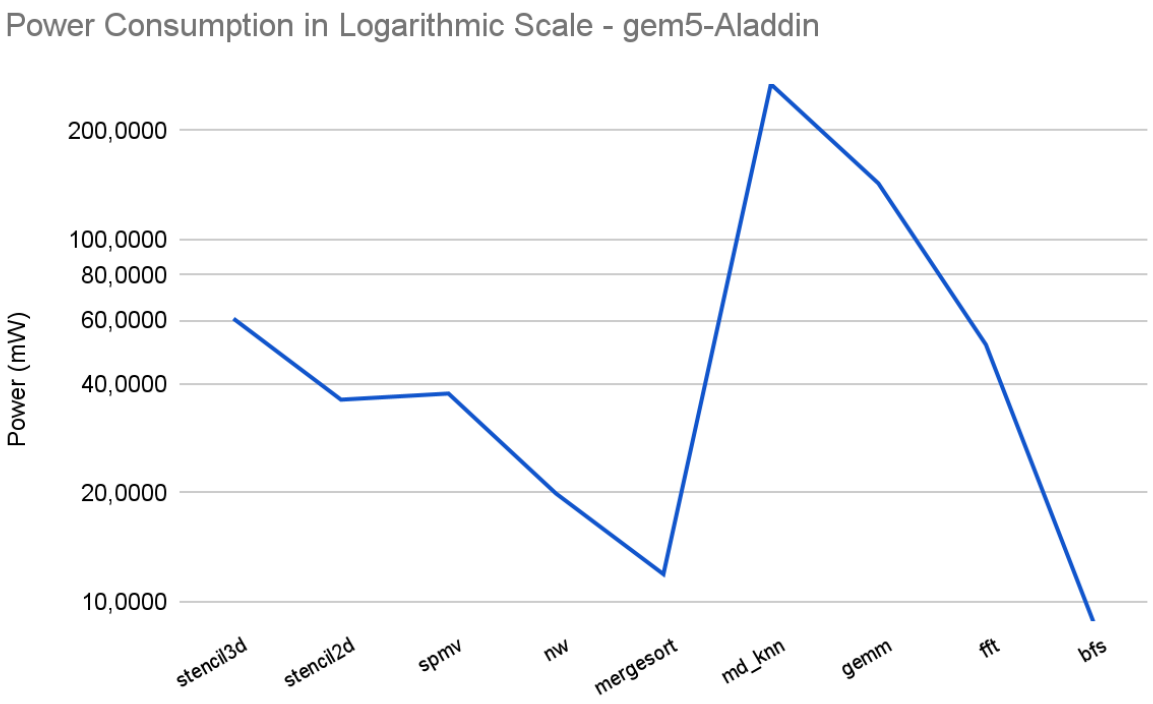


Figure 14: Power consumption for gem5-Aladdin in logarithmic scale

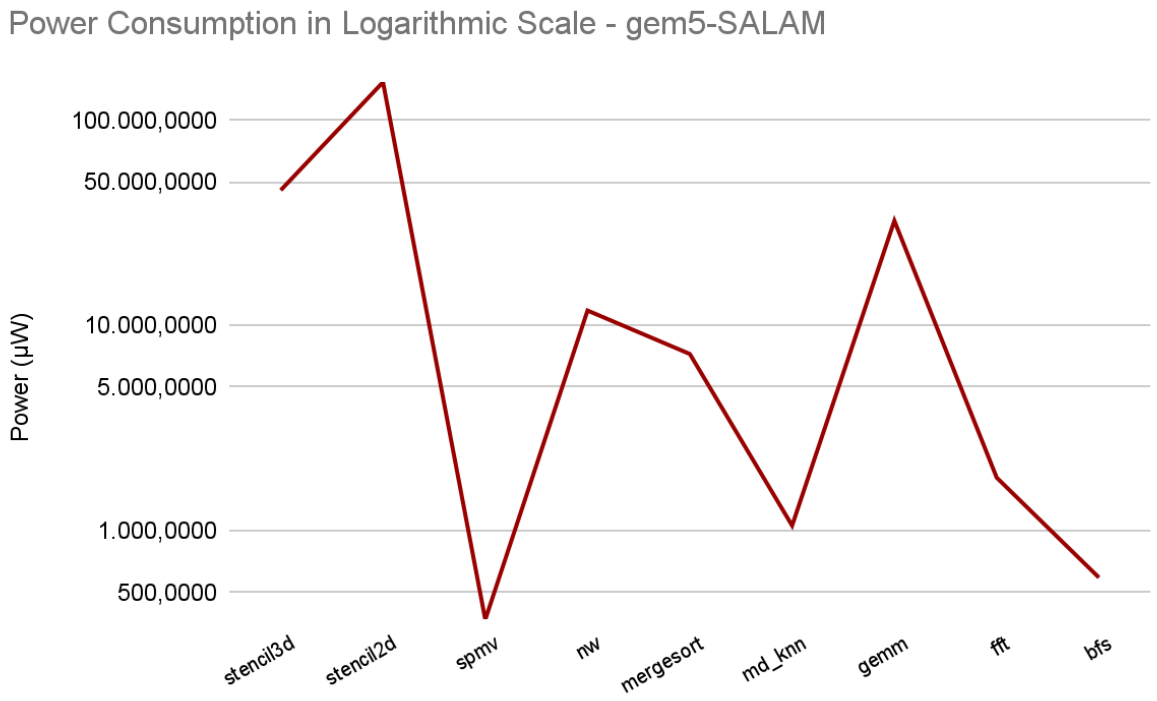


Figure 15: Power consumption for gem5-SALAM in logarithmic scale

Figures 16 & 17 show the area usage for gem5-Aladdin and gem5-SALAM across benchmarks. As with power consumption, these frameworks use different architectures and

methodologies, so this is not a direct comparison. However, the figures highlight a significant difference, with gem5-Aladdin showing much larger area usage than gem5-SALAM, indicating potential differences in area computation.

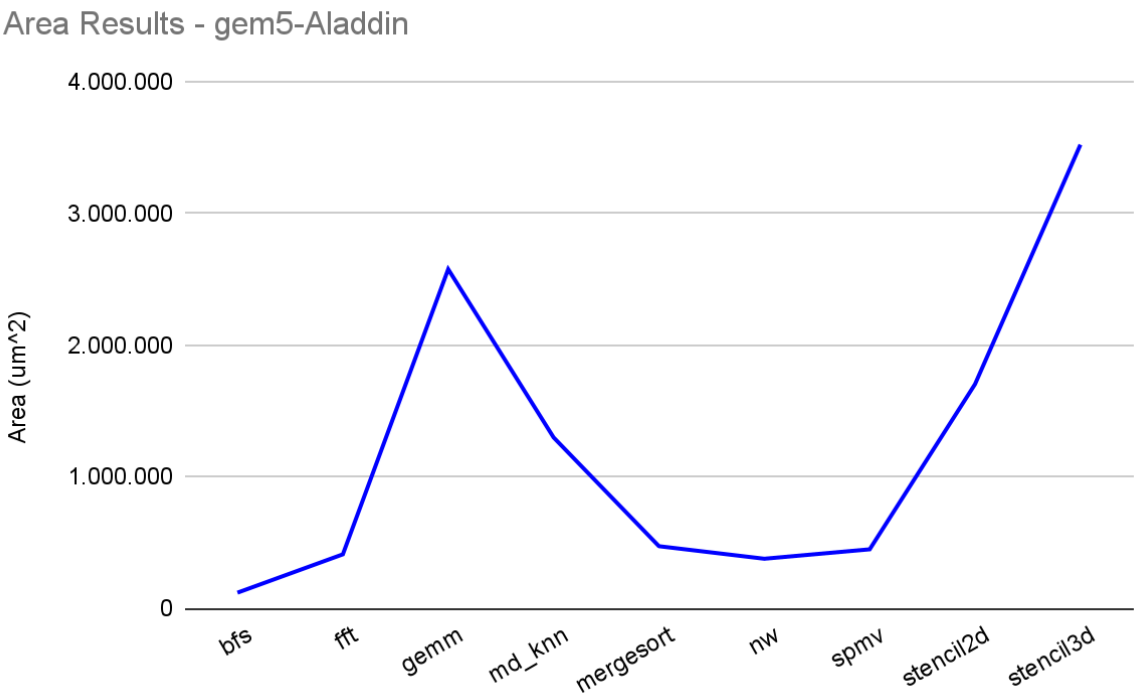


Figure 16: Area results of gem5-Aladdin across benchmarks

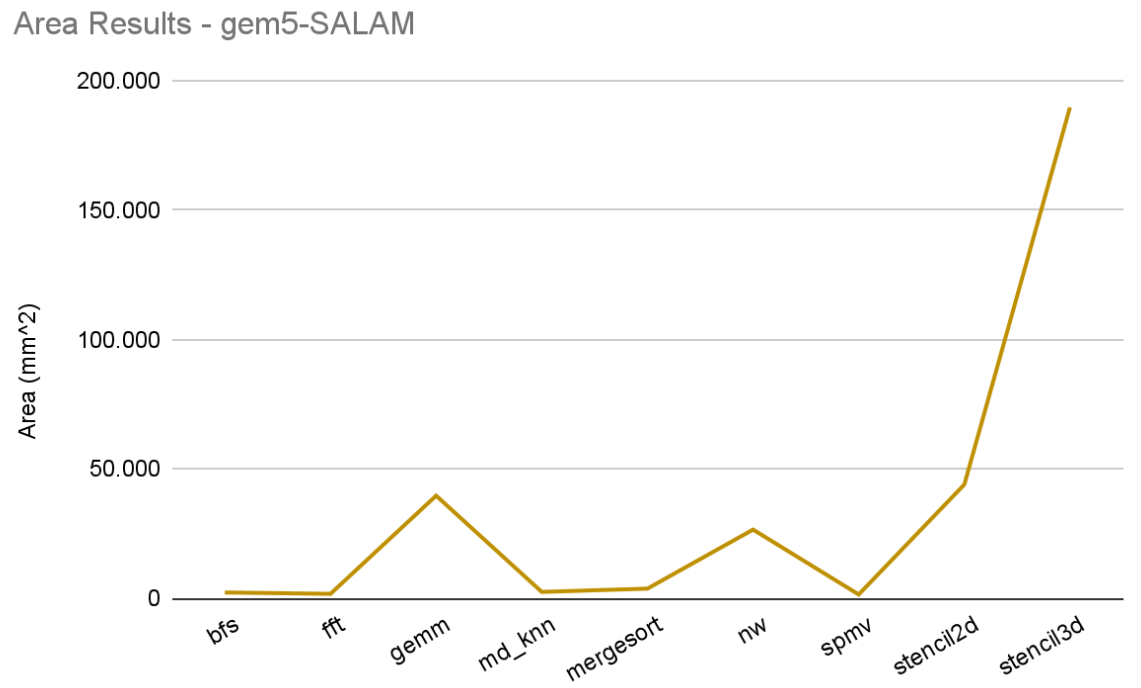


Figure 17: Area results of gem5-SALAM across benchmarks

ii) Comparison results for CPU and accelerator

Figure 18 shows the total clock cycles for each benchmark, comparing gem5-Aladdin and gem5-SALAM. It is observed that gem5-SALAM has significantly more cycles in the stencil2d and stencil3d benchmarks compared to gem5-aladdin, while all other benchmarks appear to have similar cycle counts.

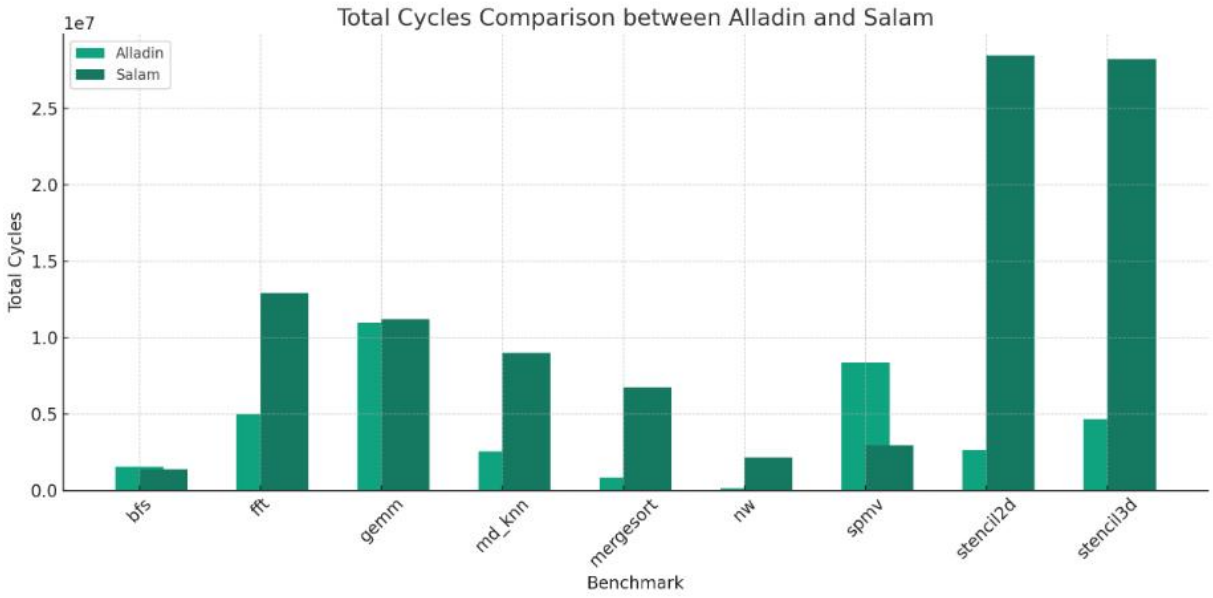


Figure 18: Total cycles comparison between gem5-Aladdin and gem5-SALAM

Figure 19 shows the number of instructions for gem5-Aladdin and gem5-SALAM across these benchmarks, there is a noticeable difference in the instructions for the bfs and spmv benchmarks, which are larger in gem5-aladdin compared to gem5-SALAM. As for the nw benchmark, the behavior is opposite, with instructions in gem5-aladdin being significantly smaller.

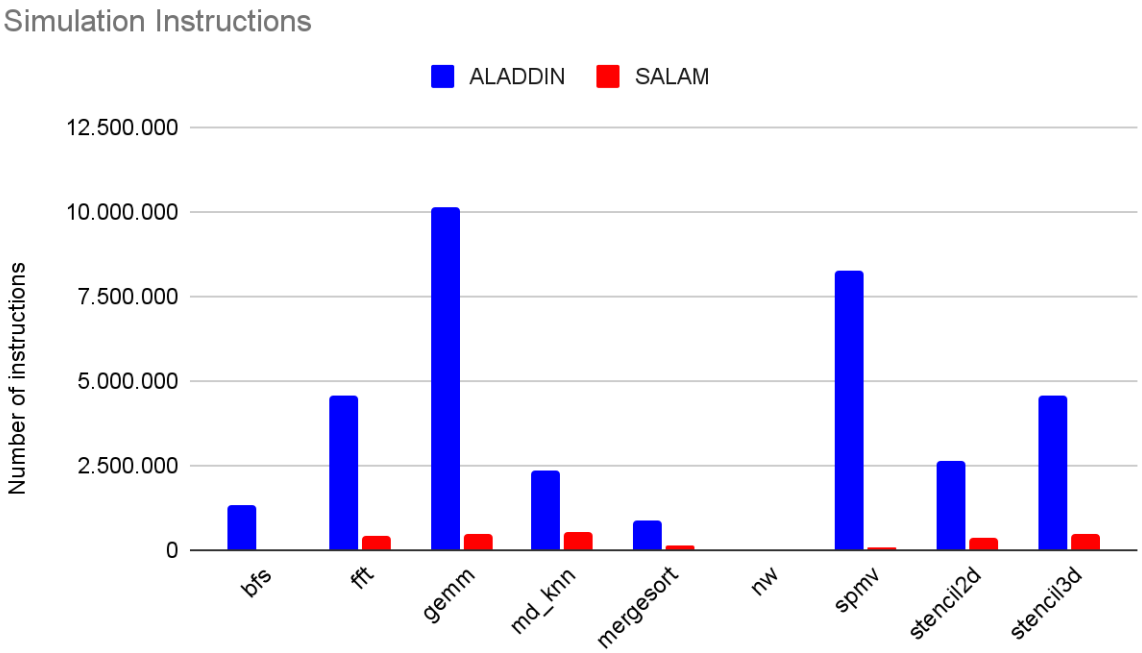


Figure 19: Comparison of instructions for gem5-Aladdin vs gem5-SALAM across benchmarks

Figure 20 is comparing the total simulation time for gem5-Aladdin and gem5-SALAM across the nine benchmarks. This figure exhibits similar characteristics to the Figure 18, as simulation time and clock cycles are correlated.

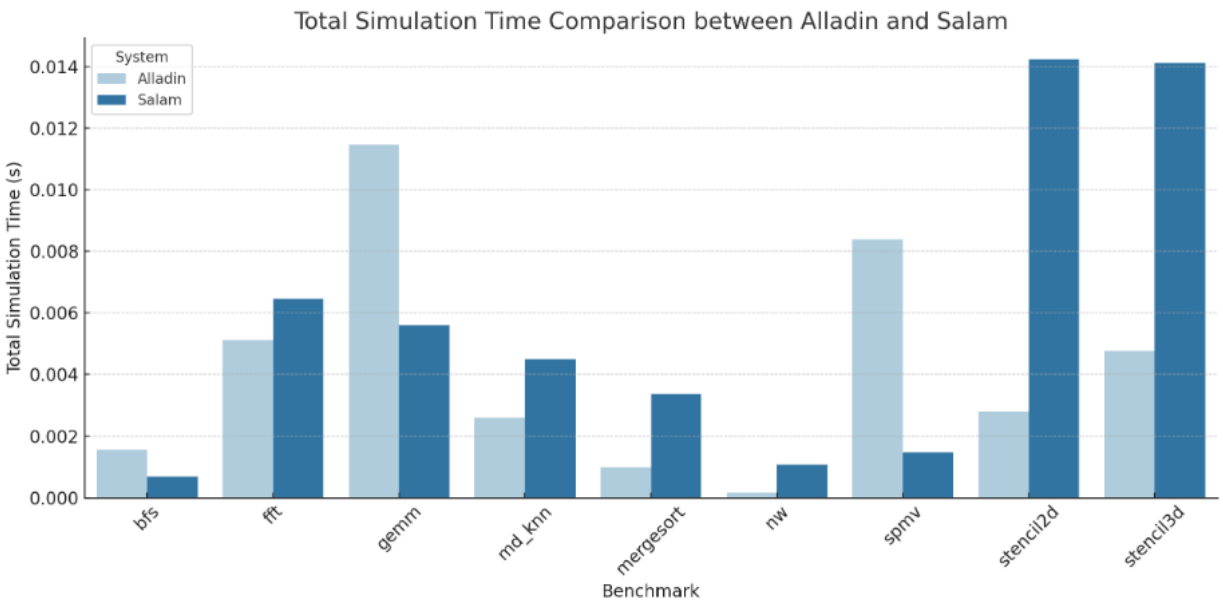


Figure 20: Total simulation time comparison between gem5-Aladdin and gem5-SALAM

iii) gem5-ALADDIN: comparison CPU VS CPU and ACC

Figure 21 shows the total clock cycles for CPU only and for the CPU with an accelerator (ACC) across the benchmarks. It is observed that there is a significant difference in CPU cycles in the fft and stencil3d benchmarks, where they are higher compared to the cycles of the accelerator and CPU. Additionally, it is noted that the gemm benchmark could not be compared due to a runtime error in gem5-aladdin.

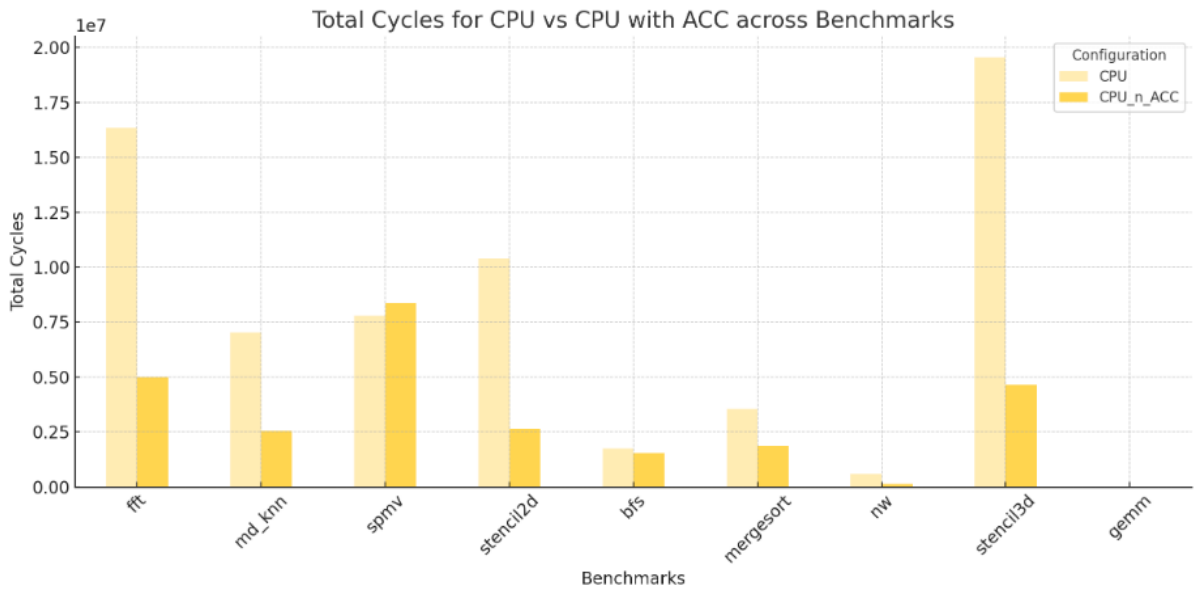


Figure 21: Total cycles for CPU vs CPU with accelerator across benchmarks

Figure 22 below compares the number of instructions for CPU and ACC and CPU across different benchmarks. There is a significant difference depicted in the stencil3d benchmark, where instructions are much higher on the CPU, while conversely, in the spmv benchmark, the opposite trend is observed.

Figure 23, also, is visualizing the simulation time for these benchmarks using either the CPU alone or a combination of CPU and ACC. It presents similar comparative results to Figure 7, despite having a different design, as simulation time and cycles are proportional.

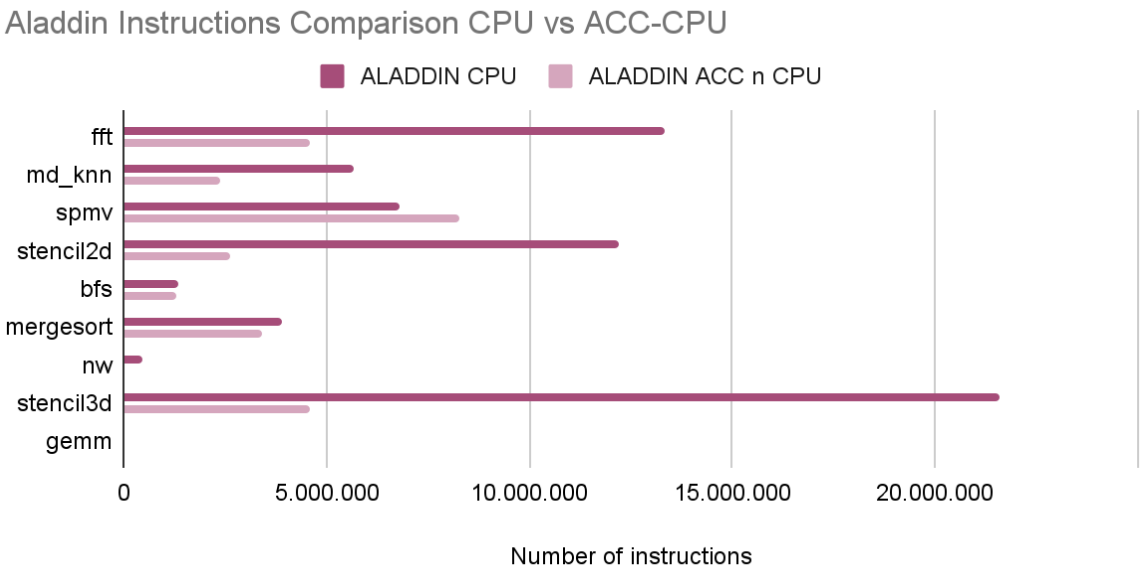


Figure 22: Instructions comparison between CPU and accelerator vs CPU only

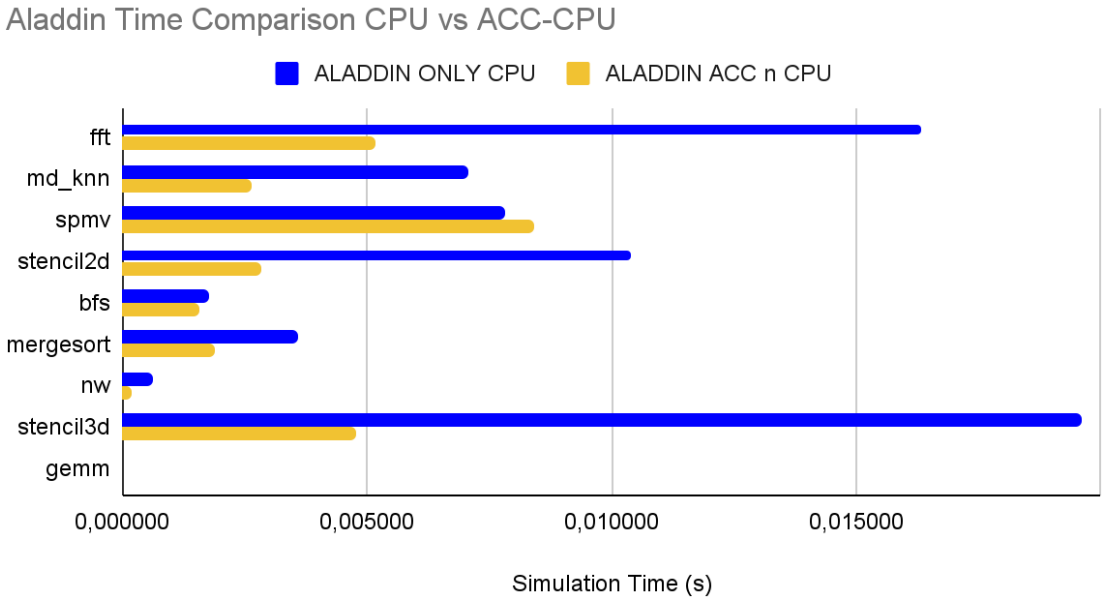


Figure 23: Simulation time comparison CPU vs CPU & Accelerator

iii) gem5-SALAM: comparison CPU vs CPU and ACC

Figure 24 compares the total clock cycles for CPU and CPU and ACC across different benchmarks. Figure 25, on the other hand, shows the comparison of executed instructions for CPU versus CPU and ACC across these benchmarks. In both figures, there is a relatively significant difference observed in the measurements of the gemm benchmark when running the program only on the CPU, where it exhibits higher cycles and instructions.

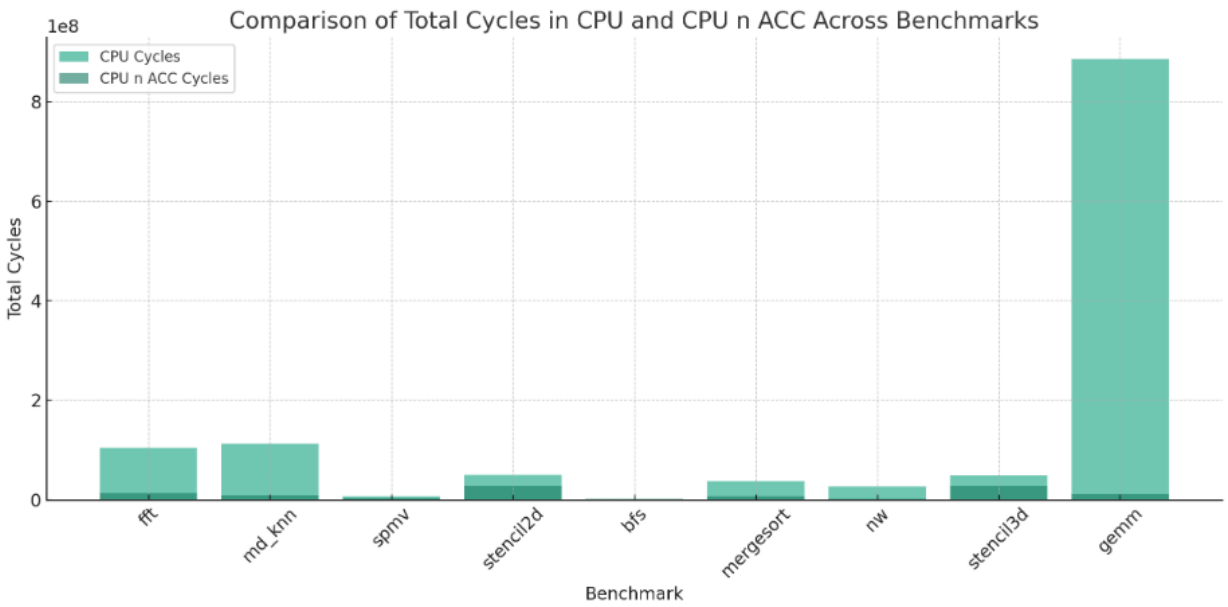


Figure 24: Comparison of total cycles in CPU only and CPU and accelerator across benchmarks

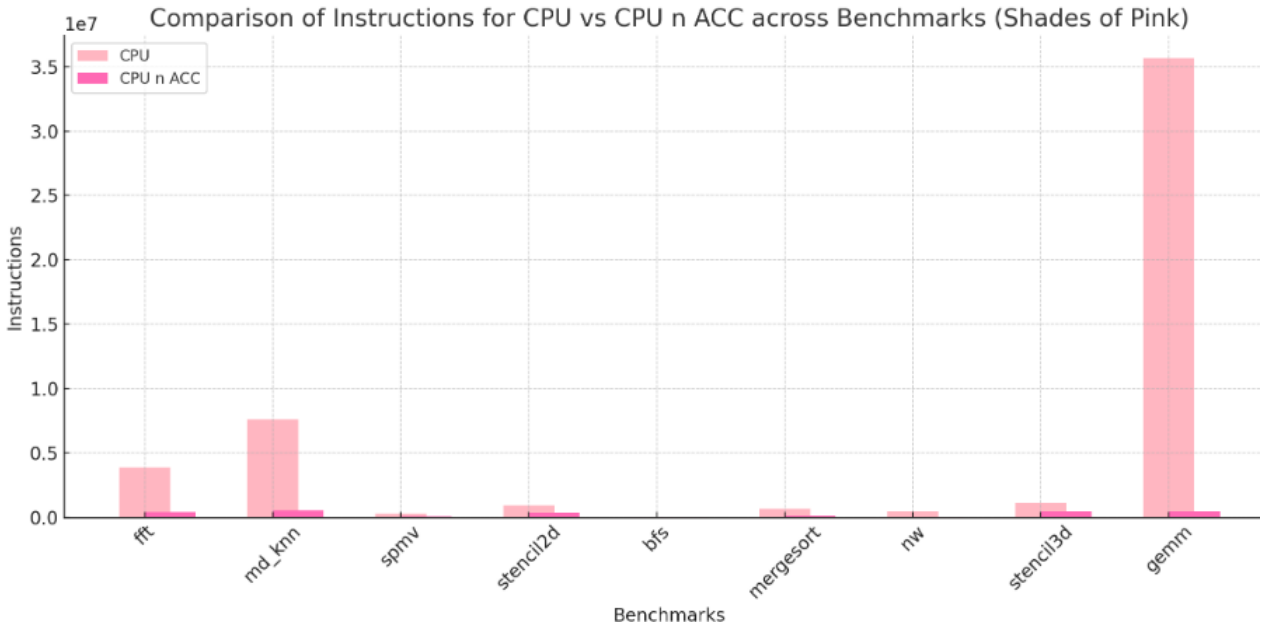


Figure 25: Comparison of instructions for CPU vs CPU and accelerator across benchmarks

Figure 26 is comparing the simulation times for CPU and CPU with ACC across the different benchmarks, having similar results as the above figures.

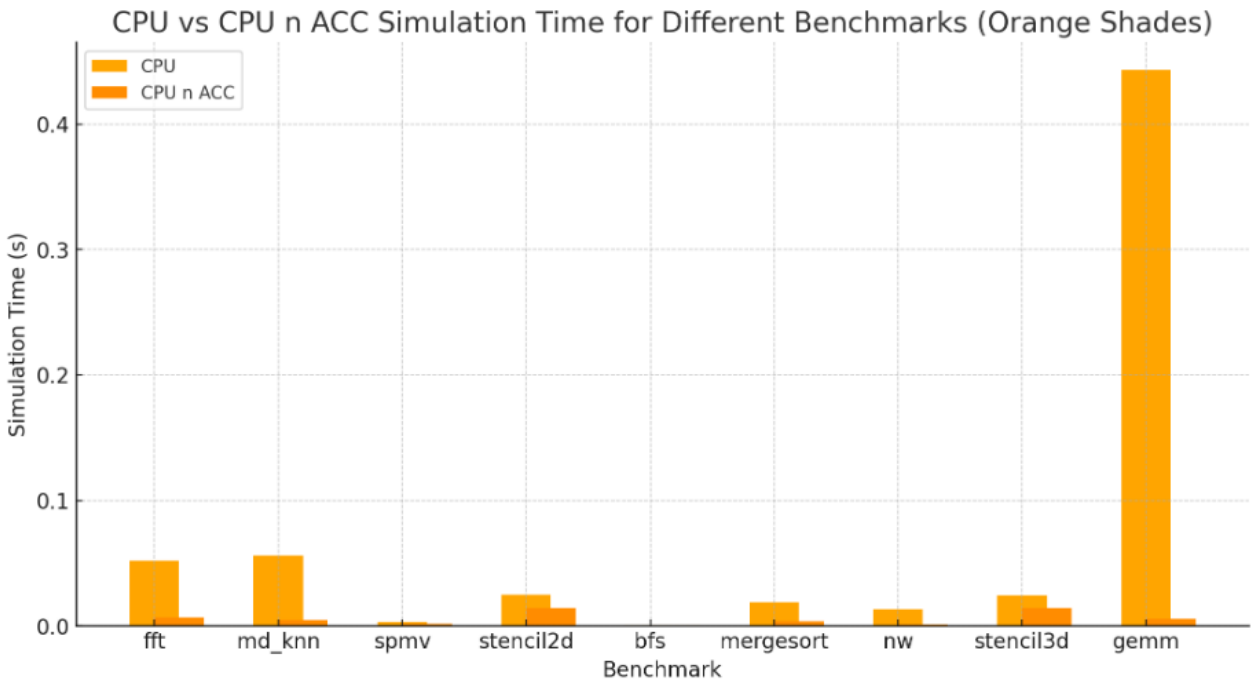


Figure 26: CPU vs CPU with accelerator simulation time for different benchmarks

iv) Comparison CPU x86 vs ARM

Figure 27 is a visual representation of the total clock cycles for each benchmark comparing x86 and ARM architectures. Here, it appears that the x86 architecture exhibits slightly higher cycles across all benchmarks. Additionally, it is mentioned that the bfs benchmark could not be compared due to a runtime error in both frameworks.

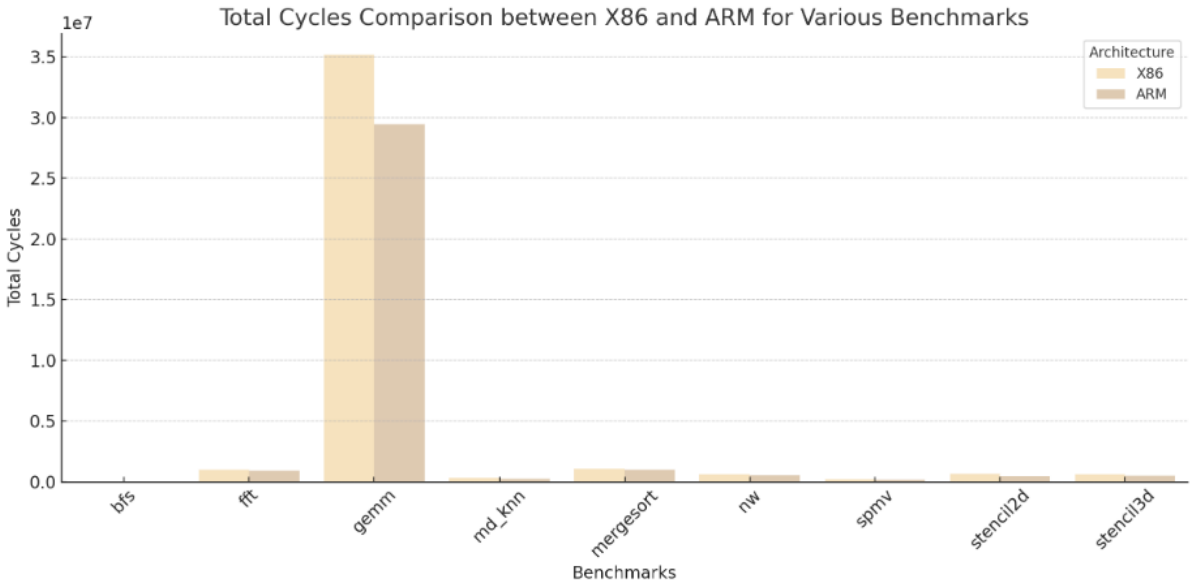


Figure 27: Total cycles comparison between X86 and ARM for various benchmarks

Figure 28 and Figure 29 compares the instruction counts and number of loaded instructions respectively between x86 and ARM architectures across these benchmarks. The same behavior is observed as in the previous figure, with the standout benchmark being gemm, which exhibits a large volume of instructions.

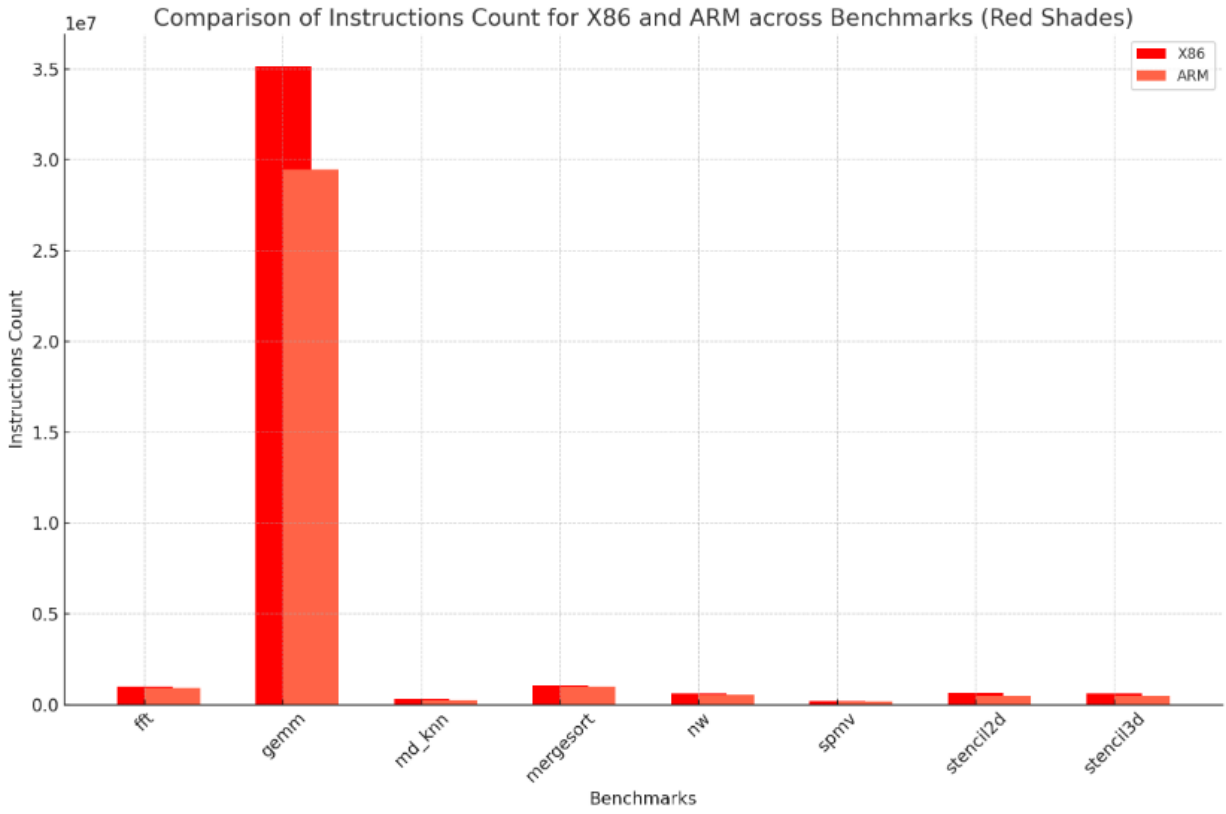


Figure 28: Comparison of instructions count for X86 and ARM across benchmarks

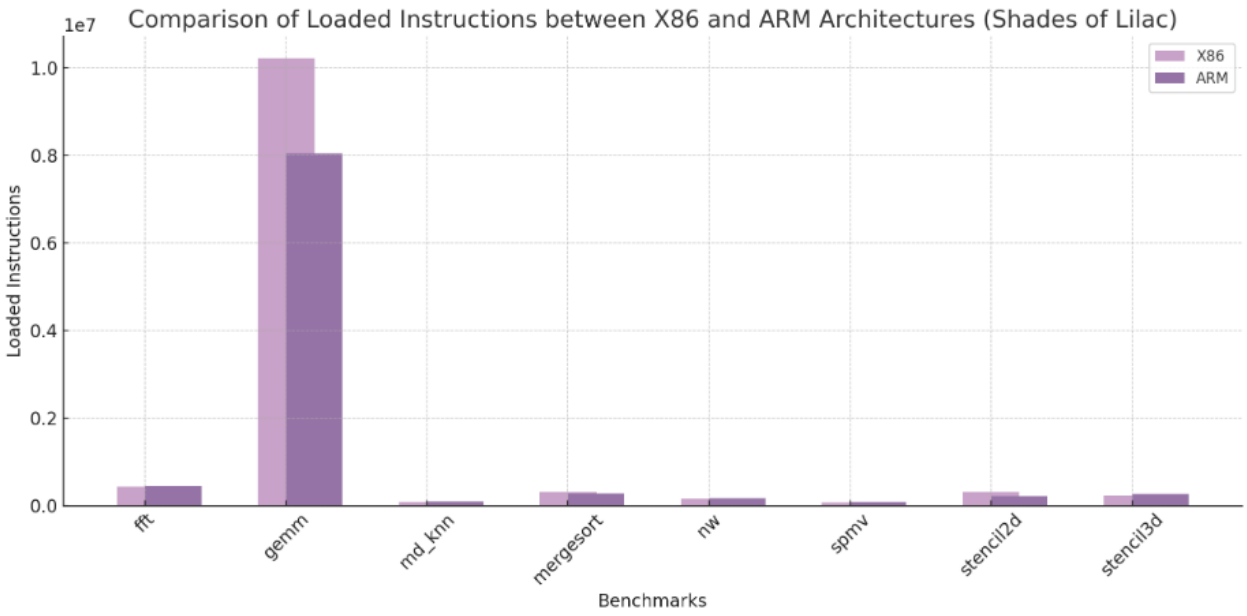


Figure 29: Comparison of loaded instructions between X86 and ARM acrhitectures

Lastly, Figure 30 is a visualization of the total simulation time comparisons between x86 and ARM architectures for the various benchmarks. Similarly, this also exhibits comparable behavior to the previous figures, with gemm standing out for its significant volume of simulation time.

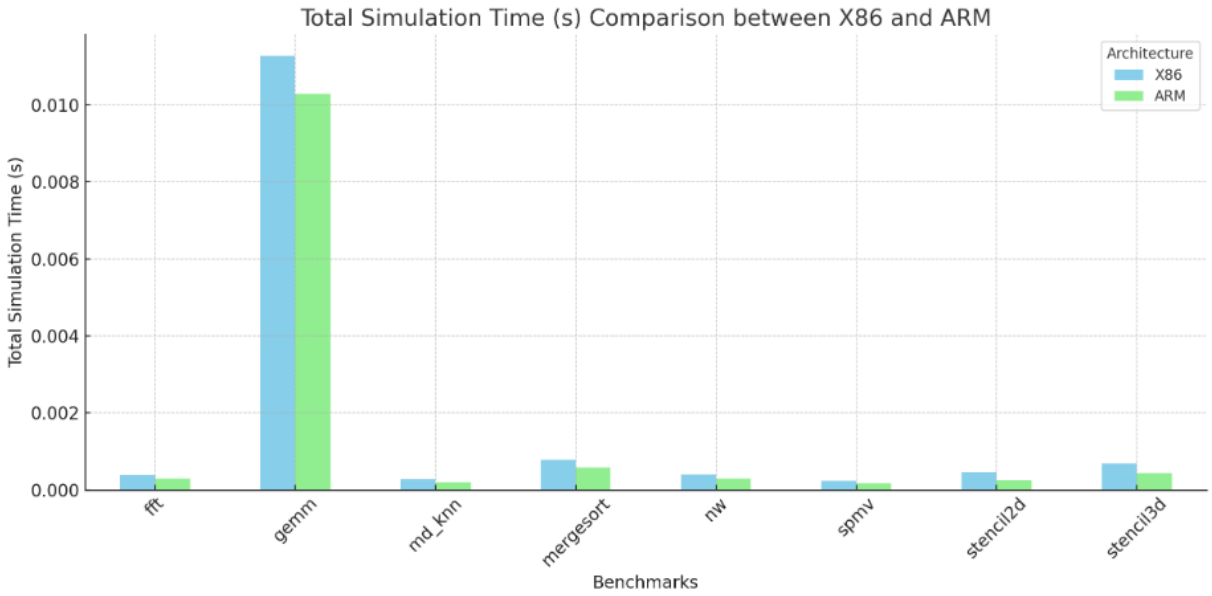


Figure 30: Total simulation time comparison between X86 and ARM

5.3 Interpretation of results, including performance metrics (cycles, commands, times)

In the evolving landscape of computational accelerators, the evaluation and comparison of performance metrics such as accelerator cycles, power consumption, area efficiency, instruction counts, and total simulation time are fundamental in assessing system efficiency

and architectural effectiveness. A meticulous analysis of these metrics across diverse benchmarks reveals profound insights into the operational characteristics and optimization potential of two prominent systems: gem5-ALADDIN and gem5-SALAM.

Accelerator Cycles:

A critical aspect of our analysis involves the assessment of accelerator cycles across different benchmarks. Predominantly, gem5-ALADDIN demonstrates a higher cycle count compared to gem5-SALAM, with notable exceptions in md_knn and nw benchmarks. This variation can be attributed to the distinct computational demands and architectural compatibilities of these benchmarks. Specifically, md_knn, characterized by its compute-intensive nature, benefits from DMA's capability to efficiently overlap computation with data movement. On the other hand, nw, with its regular access patterns and minimal data prerequisites for computation, aligns more favorably with a cache-based memory system devoid of DMA. These observations suggest a nuanced interplay between accelerator design and benchmark characteristics, impacting overall cycle efficiency.

Power and Area:

The analysis of power usage and area metrics uncovers coding and architectural differences, particularly in gem5-SALAM. gem5-ALADDIN employs a power calculation formula that combines average functional unit power with average memory power, incorporating both dynamic and leakage power components. In contrast, gem5-SALAM's power consumption computation adopts a distinct approach, focusing on static and dynamic power elements across various functional units such as adders, multipliers, and registers. This divergence in power consumption computation methodologies underscores the inherent architectural differences between the two systems and their implications on power and area efficiency.

Total Clock Cycles and Instructions:

An examination of total cycles and instruction counts across benchmarks highlights architectural influences and system execution modes. In most cases, gem5-SALAM registers a higher cycle count, which can be ascribed to differences in architecture, instruction set architecture (ISA), and particularly, the system execution mode. Exceptions like bfs, with similar cycle counts in both systems, and spmv, characterized by its indirect memory access patterns, further reinforce the impact of architectural factors. Notably, gem5-ALADDIN records a higher instruction count in almost all benchmarks, indicative of its architectural design where the CPU handles fewer instructions, offloading the majority to the accelerator. The exception of nw, where gem5-ALADDIN demonstrates marginally fewer instructions, possibly points to internal management efficiencies.

Total Simulation Time:

The comparison of total simulation time reveals that gem5-ALADDIN does not parallel gem5-SALAM in cycle efficiency, despite analogous operational roles. This discrepancy suggests potential inefficiencies or constraints within gem5-ALADDIN's system. In scenarios involving both CPU and accelerator, gem5-ALADDIN exhibits increased cycles, particularly in spmv, implying augmented communication demands for memory access.

CPU vs. Accelerator Performance:

Transitioning from CPU-only to CPU-accelerator configurations, a significant rise in cycles is observed for CPU-only operations. This increase is rationalized by the additional workload undertaken by the CPU in the absence of an accelerator, with the exception of spmv. Here, the increased communication overhead for memory access appears to counterintuitively create more cycles. Instruction counts predominantly escalate in the absence of the accelerator, barring spmv, due to its unique memory access pattern. Correspondingly, simulation times increase, mirroring the trend in total cycles and instructions.

ARM vs. x86 Architecture:

The juxtaposition of ARM and x86 architectures in this analysis illuminates their distinct operational methodologies. ARM's concise approach starkly contrasts with the extensive nature of x86, influencing overall system performance and architectural efficiency.

Benchmark-Specific Observations:

md_knn: This benchmark registers the highest power discrepancy, likely due to a heavier reliance on muxes and variable non-arithmetic operators. The variability of these operators leads to an overestimation of power requirements.

spmv: Challenges in runtime loop dependencies, limitations in static loop unrolling, and constraints on runtime parallelism are notable. These factors contribute to constrained performance and efficiency in spmv.

The benchmark executables, derived from gem5-SALAM and slightly modified for each architecture, were executed in gem5's system emulation mode. This mode, being more direct than the full system mode, potentially accounts for the lower cycle counts observed in both architectures. The full system mode, offering a comprehensive simulation environment, naturally involves extended simulation times but provides a richer context for system-level behavior analysis.

The comparative analysis of gem5-ALADDIN and gem5-SALAM across a spectrum of benchmarks provides a multi-dimensional view of their performance dynamics. This study not only highlights the strengths and limitations inherent in each system but also paves the way for future enhancements in design and optimization strategies within the domain of high-performance computing systems. The findings underscore the critical role of system architecture, memory management, and accelerator integration in shaping computational efficiency and performance.

5.4 Comparison of gem5-SALAM and gem5-Aladdin based on the common benchmarks

In the landscape of high-performance computing, the evaluation of gem5-ALADDIN and gem5-SALAM across a spectrum of benchmarks provides insightful contrasts in their operational efficiencies. This analysis, encompassing nine distinct benchmarks, examines their performance variations in terms of accelerator cycles, power consumption, and area. In the case of bfs (Breadth-First Search), gem5-ALADDIN tends to exhibit a higher cycle count, possibly due to a less efficient data traversal mechanism. This higher operational intensity often translates to increased power consumption, although gem5-SALAM might demonstrate greater area efficiency, reflecting a more compact design suitable for graph traversal algorithms. For fft (Fast Fourier Transform), a computationally intensive benchmark, gem5-ALADDIN may require more cycles, indicating a less efficient implementation of the transform algorithm. This increased computational demand is likely to result in higher power usage. Conversely, gem5-SALAM could show a more area-efficient design for fft, benefiting from optimized hardware suited for such complex transformations.

In the realm of GEMM (General Matrix Multiply), a key operation in scientific and engineering applications, gem5-SALAM can achieve greater cycle efficiency by pipelining loops and optimizing the system with static, hard-coded memory accesses. In contrast, gem5-ALADDIN may consume more power due to its potentially higher operational intensity in matrix operations. Area-wise, gem5-SALAM might exhibit lower usage, suggesting a more efficient layout for matrix computations. For md_knn (Molecular Dynamics K-Nearest Neighbors), gem5-SALAM might display lower cycle counts, benefiting from efficient data movement and compute overlap. gem5-ALADDIN, on the other hand, could show higher power consumption due to its extensive computational and data handling requirements. In terms of area, gem5-SALAM may have the edge, with a design more attuned to the demands of molecular dynamics simulations.

The nw (Needleman-Wunsch) benchmark is particularly interesting, as it might demonstrate lower cycles in gem5-ALADDIN, suggesting better handling of regular access patterns and smaller data sets. This efficiency might not extend to power usage, where gem5-SALAM could be more efficient, especially if it manages to execute the algorithm with fewer cycles. Area-wise, gem5-ALADDIN might be less efficient, given its potentially higher cycle counts and operational demands. In mergesort, gem5-ALADDIN may demonstrate higher cycle counts, possibly due to less efficient sorting algorithms or memory access patterns. The

power consumption in mergesort might be higher for gem5-ALADDIN, aligning with its greater computational demand. gem5-SALAM could exhibit a more area-efficient design for mergesort, benefiting from an optimized approach to sorting algorithms.

The spmv (Sparse Matrix-Vector Multiplication) benchmark could show gem5-ALADDIN with a higher cycle count, possibly due to less efficient handling of sparse data structures. Power usage in spmv might be more for gem5-ALADDIN, correlating with its higher operational intensity in processing sparse matrices. gem5-SALAM might demonstrate better area efficiency in spmv, potentially due to a more compact layout tailored for sparse data operations.

In stencil2d and stencil3d, gem5-ALADDIN might exhibit higher numbers of cycles, indicative of less optimized data grid manipulation. These benchmarks might also see gem5-ALADDIN consuming more power usage, reflecting its more intensive computation. In terms of area efficiency, gem5-SALAM may have the advantage, with a possibly more optimized design for both 2D and 3D grid computations.

Overall, the comparative analysis across these benchmarks reveals gem5-ALADDIN's tendency towards higher cycle counts and power usage, suggesting a design favoring computational robustness. gem5-SALAM, in contrast, appears more cycle-efficient and conserves power better, likely due to a balanced design optimizing resource utilization. These distinctions highlight the need for careful consideration of computational requirements, resource constraints, and efficiency goals when choosing between these systems for specific applications in high-performance computing.

6. CONCLUSION

6.1 Overall conclusion

In this thesis, we analyzed various gem5-based and gem5-compatible simulation frameworks, including gem5-Aladdin, Smaug, gem5-SALAM, gem5-X, and MAESTRO, to evaluate their effectiveness in modeling heterogeneous SoC designs with domain-specific accelerators. We assessed these frameworks based on their simulation speed, overall performance, ease of customization, and ease of use, while also exploring how different settings and modeling methods in gem5 impact the integration of accelerators into systems. This analysis provides valuable insights into selecting the right tools for optimizing accelerator design and integration in modern computing systems.

In the final stage of this research, we compared gem5-SALAM and gem5-Aladdin to demonstrate how accelerators can be effectively simulated within a gem5 environment. While both frameworks offer strong solutions for SoC evaluation, gem5-SALAM is more suited for detailed architectural exploration due to its configurability, whereas gem5-Aladdin is better for rapid prototyping with its user-friendly interface. This comparison highlights the strengths and limitations of each tool, offering practical guidance for selecting the appropriate framework for specific needs in accelerator simulation and SoC design.

6.2 Summary of findings from the comparison

The in-depth comparison between gem5-ALADDIN and gem5-SALAM, across a range of benchmarks in high-performance computing, has yielded critical insights into their operational characteristics and efficiencies. This analysis reveals distinct differences in their performance, painting a comprehensive picture of their respective strengths and limitations.

gem5-ALADDIN, in most benchmarks, exhibits a higher count of accelerator cycles. This trend suggests an operational intensity and robustness in handling computationally demanding tasks. This characteristic of gem5-ALADDIN is particularly evident in benchmarks that require intensive data processing or complex computations, where it demonstrates its prowess in managing challenging tasks. However, this increased operational intensity often translates to higher power consumption, a factor that needs to be considered in energy-sensitive applications.

On the other hand, gem5-SALAM generally shows lower cycle counts across these benchmarks, indicating a more efficient approach to handling computational tasks. This efficiency in gem5-SALAM's operation can be attributed to its optimized data management strategies and balanced integration of processing and accelerator utilization. The lower power consumption observed in gem5-SALAM across various benchmarks reflects an architecture that prioritizes energy efficiency, an essential aspect in scenarios where power constraints are critical.

When it comes to area efficiency, gem5-SALAM often has the upper hand, exhibiting a more compact and space-efficient design. This characteristic makes gem5-SALAM particularly

advantageous in applications where physical space and layout efficiency are significant considerations. In contrast, gem5-ALADDIN, while formidable in its computational capabilities, may not be as area-efficient, potentially necessitating more physical space for equivalent computational tasks.

The analysis also reveals that gem5-ALADDIN typically has a higher instruction count, suggesting a design where the CPU is less burdened, delegating more tasks to the accelerator. This trend indicates gem5-ALADDIN's reliance on accelerator-driven computations, a key feature in its architectural design. gem5-SALAM's architecture, which results in a lower instruction count, points to a more effective distribution of tasks between the CPU and the accelerator, highlighting its efficient operational model.

Certain benchmarks, such as md_knn and nw, bring to light scenarios where gem5-ALADDIN outperforms gem5-SALAM, or vice versa. These instances underscore the importance of choosing a system that aligns with the specific computational requirements of the application at hand.

The findings from this comparative study emphasize the need to consider the specific requirements of the intended application when selecting between gem5-ALADDIN and gem5-SALAM. gem5-ALADDIN's strength in handling computationally intensive tasks makes it suitable for scenarios requiring robust processing power. Conversely, gem5-SALAM, with its focus on efficiency and resource optimization, is ideal for applications where energy conservation and spatial constraints are paramount.

In conclusion, the decision to opt for either gem5-ALADDIN or gem5-SALAM should be informed by the nature of the computational tasks, the intensity of the operations, and the resource constraints of the application environment. This study provides valuable guidance for system architects and developers, aiding in the selection and optimization of computational systems for diverse high-performance computing applications.

7. FUTURE WORK

Evaluating the future of gem5-based frameworks for heterogeneous System-on-Chip (SoC) designs reveals that the field of computing is poised for significant evolution. This chapter aims to explore potential research trajectories and development paths that could profoundly influence the landscape of computational simulations and SoC design.

Firstly, enhancing the capabilities of all gem5-based frameworks, including gem5-Aladdin, Smaug, gem5-SALAM, gem5-X, and MAESTRO, is a critical area of focus. With the increasing complexity of hardware accelerators, there is a pressing need for more sophisticated models of power consumption, thermal behavior, and precise timing. Developing advanced power and thermal models within these frameworks would elevate the accuracy of predictions and cater to the stringent efficiency demands of energy-sensitive applications. Additionally, refining timing models to more accurately reflect real-time operations could significantly benefit critical systems, such as autonomous vehicles and healthcare devices.

Moving beyond traditional boundaries, the integration of emerging technologies presents an exciting frontier. The transformative potential of incorporating quantum computing elements into SoC designs could greatly enhance computational capabilities, necessitating the evolution of simulation frameworks to effectively model these groundbreaking components. Similarly, integrating AI and machine learning algorithms could revolutionize simulation processes, automating and optimizing workflows to lead to more efficient design cycles and robust predictive analytics.

Scalability and adaptability are also pivotal aspects. As computational demands escalate, the ability of simulation frameworks to scale becomes increasingly crucial. Developing cloud-based versions of these frameworks would offer scalable resources for complex simulations, providing researchers with access to high-performance computing environments. Ensuring cross-platform compatibility would guarantee applicability across diverse SoC environments, broadening the utility and impact of these tools. Furthermore, improving user interfaces is essential; intuitive graphical user interfaces (GUIs) for all frameworks would make them more accessible, especially for those unfamiliar with command-line interfaces. Interactive simulation tools that allow for real-time modifications could significantly enhance the design and evaluation experience.

The role of collaborative and open-source initiatives cannot be overstated. By fostering a community-driven development approach, the global research community can contribute to the enhancement of these frameworks, leading to more robust and feature-rich tools. Additionally, forging partnerships with industry leaders can provide invaluable practical insights into real-world applications, driving the development of industry-relevant features and capabilities. Comprehensive benchmarking and the standardization of evaluating protocols are also vital. The development of new benchmarks that reflect the latest trends in computing, such as edge computing and IoT devices, is essential. Moreover,

standardizing evaluation protocols across different frameworks would enable more meaningful comparisons and informed decision-making processes. Finally, looking towards a long-term vision, the ultimate goal could be the realization of autonomous SoC design systems. This would involve leveraging machine learning to create self-optimizing systems capable of adapting their architecture and operations based on real-time data and performance metrics. It also includes predictive modeling for future technologies, anticipating technological advancements, and preemptively creating models and tools that can accommodate these innovations.

In conclusion, the realm of gem5-based frameworks for SoC design and simulation stands at a pivotal juncture. The future work in this field promises not only enhancements to existing methodologies but also the opening of doors to revolutionary concepts in computing architecture. As these frameworks are further advanced, their development will be crucial for keeping pace with the rapidly growing computational demands of our technologically driven modern world.

REFERENCES

[1] Bhagwat, P., & Tripathi, S. K. (1995). Mobile Computing: A research perspective. In Computer Networks, Architecture and Applications: Proceedings of the IFIP TC6 conference 1994 (pp. 3-12). Springer US.

[2] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., ... & Hestness, J. (2011). The gem5 Simulator. ACM SIGARCH Computer Architecture News, 39(2), 1-7.

[3] Bolhasani, H., & Jassbi, S. J. (2020). Deep learning accelerators: a case study with MAESTRO. Journal of Big Data, 7, 1-11.

[4] Burgazzi, L., & Pierini, P. (2007). Reliability studies of a high-power proton accelerator for accelerator-driven system applications for nuclear waste transmutation. Reliability Engineering & System Safety, 92(4), 449-463.

[5] Chakrapani, L. N., Akgul, B. E., Cheemalavagu, S., Korkmaz, P., Palem, K. V., & Seshasayee, B. (2006, March). Ultra-efficient (embedded) SOC architectures based on probabilistic CMOS (PCMOS) technology. In Proceedings of the Design Automation & Test in Europe Conference (Vol. 1, pp. 1-6). IEEE.

[6] Chang, H. (2003). SOC design methodologies. Winning the SoC Revolution: Experiences in Real Design, 21-45.Chang, H. (2003). SOC design methodologies. Winning the SoC Revolution: Experiences in Real Design, 21-45.

[7] Cong, M. T. (2023). Hardware accelerated simulation and automatic design of heterogeneous architecture (Doctoral dissertation, Université de Rennes).

[8] Cong, T., & Charot, F. (2019, October). Designing Application-Specific Heterogeneous Architectures from Performance Models. In 2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC) (pp. 265-272). IEEE.

[9] Coussy, P., Morawiec, A. (Eds.). (2008). An Introduction to High-Level Synthesis. Springer. https://doi.org/10.1007/978-1-4419-0936-1

[10] Dally, W. J., Turakhia, Y., & Han, S. (2020). Domain-specific hardware accelerators. Communications of the ACM, 63(7), 48-57.

[11] Hamidreza, B., & Jafarali, J. S. (2020). Deep learning accelerators: a case study with MAESTRO. Journal of Big Data, 7(1).

[12] Hardy, L. (2003). Accelerator reliability-availability. In KEK PROCEEDINGS (pp. 92-96). High Energy Accelerator Research Organization; 1999.

[13] Hennessy, J. L., & Patterson, D. A. (2017). Computer Architecture: A Quantitative Approach (6th ed.). Morgan Kaufmann.

[14] Hill, M. D., & Reddi, V. J. (2021). Accelerator-level parallelism. Communications of the ACM, 64(12), 36-38.

[15] Hill, M., & Reddi, V. J. (2019, February). Gables: A roofline model for mobile socs. In 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA) (pp. 317-330). IEEE.

[16] Jouppi, N. P., Young, C., Patil, N., & Patterson, D. (2017). A Domain-specific Architecture for Deep Neural Networks. Communications of the ACM, 61(9), 50-59.

[17] Kwon, H., Chatarasi, P., Pellauer, M., Parashar, A., Sarkar, V., & Krishna, T. (2019, October). Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach using MAESTRO. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (pp. 754-768).

[18] Lahiri, K., Raghunathan, A., & Dey, S. (2001, January). Evaluation of the traffic-performance characteristics of system-on-chip communication architectures. In VLSI Design 2001. Fourteenth International Conference on VLSI Design (pp. 29-35). IEEE.

[19] Leng, J., Buyuktosunoglu, A., Bertran, R., Bose, P., & Reddi, V. J. (2019). Asymmetric resilience for accelerator-rich systems. IEEE Computer Architecture Letters, 18(1), 83-86.

[20] Lüdeke, A., Hardy, L., & Giachino, R. (2014, July). The accelerator reliability forum. In Proc. 5th international particle accelerator conference (IPAC), ISBN (pp. 978-3).

[21] Malița, M., & Ștefan, G. M. A Recursive Hierarchy for Accelerator-Level Parallelism.

[22] Moura, S. J., Chaturvedi, N. A., & Krstic, M. (2012, June). PDE estimation techniques for advanced battery management systems—Part I: SOC estimation. In 2012 American Control Conference (ACC) (pp. 559-565). IEEE.

[23] Nejatollahi, H., Dutt, N., Banerjee, I., & Cammarota, R. (2018). Domain-specific accelerators for ideal lattice-based public key protocols. Cryptology ePrint Archive.

[24] Qureshi, Y. M., Simon, W. A., Zapater, M., Atienza, D., & Olcoz, K. (2019). Gem5-X: A Gem5-based system level simulation framework to optimize many-core platforms. In 2019 Spring Simulation Conference (SpringSim) (pp. 1-12). IEEE.

[25] Qureshi, Y. M., Simon, W. A., Zapater, M., Olcoz, K., & Atienza, D. (2021). Gem5-x: A many-core heterogeneous simulation platform for architectural exploration and optimization. ACM Transactions on Architecture and Code Optimization (TACO), 18(4), 1-27.

[26] Raghunathan, V., Srivastava, M. B., & Gupta, R. K. (2003, June). A survey of techniques for energy efficient on-chip communication. In Proceedings of the 40th annual Design Automation Conference (pp. 900-905).

[27] Reddi, V. J., Kanter, D., Mattson, P., Duke, J., Nguyen, T., Chukka, R., ... & Zika, D. (2020). MLPerf mobile inference benchmark. arXiv preprint arXiv:2012.02328.

[28] Rogers, S., Slycord, J., Baharani, M., & Tabkhi, H. (2020, October). gem5-SALAM: A system architecture for LLVM-based accelerator modeling. In 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) (pp. 471-482). IEEE.

[29] Roofline, O. (2019). Gables: A Roofline Model for Mobile SoCs.

[30] Saleh, R., Wilton, S., Mirabbasi, S., Hu, A., Greenstreet, M., Lemieux, G., ... & Ivanov, A. (2006). System-on-chip: Reuse and integration. Proceedings of the IEEE, 94(6), 1050-1069.

[31] Shao, Y. S., & Brooks, D. (2016). gem5-Aladdin: A Pre-RTL, Power-Performance Simulator for Heterogeneous Systems. Proceedings of the 2016 International Conference on Parallel Architecture and Compilation

[32] Shao, Y. S., Xi, S. L., Srinivasan, V., Wei, G. Y., & Brooks, D. (2016, October). Co-designing accelerators and SoC interfaces using gem5-Aladdin. In 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) (pp. 1-12). IEEE.

[33] Sze, V., Chen, Y.-H., Yang, T.-J., & Emer, J. (2017). Efficient Processing of Deep Neural Networks: A Tutorial and Survey. Proceedings of the IEEE, 105(12), 2295-2329. https://doi.org/10.1109/JPROC.2017.2761740

[34] Wieferink, A., Meyr, H., Leupers, R., Wieferink, A., Meyr, H., & Leupers, R. (2008). SOC Design Methodologies. Retargetable Processor System Integration into Multi-Processor System-on-Chip Platforms, 7-23.

[35] Xi, S., Yao, Y., Bhardwaj, K., Whatmough, P., Wei, G. Y., & Brooks, D. (2020). SMAUG: End-to-end full-stack simulation infrastructure for deep learning workloads. ACM Transactions on Architecture and Code Optimization (TACO), 17(4), 1-26.

[36] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, “MachSuite: Benchmarks for Accelerator Design and Customized Architectures,” in IISWC, 2014.

[37] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, “Aladdin: A Pre-RTL, Power-Performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures,” in ISCA, 2014.

[38] Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., and Jouppi, N. P. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).* 469–480.

[39] Odysseas Chatzopoulos, George Papadimitriou, Vasileios Karakostas, and Dimitris Gizopoulos. 2024.gem5-MARVEL: Microarchitecture-Level Resilience Analysis of Heterogeneous SoC Architectures. In IEEE International Symposium on High-Performance Computer Architecture (HPCA 2024).

[40] K. T. Lim, D. Meisner, A. G. Saidi, P. Ranganathan, and T. F. Wenisch, “Thin Servers with Smart Pipes: Designing SoC Accelerators for Memcached,” in ISCA, 2013.

[41] L. Wu, R. J. Barker, M. A. Kim, and K. A. Ross, “Navigating Big Data with High-Throughput, Energy-Efficient Data Partitioning,” in ISCA, 2013.

[42] William Andrew Simon, Yasir Mahmood Qureshi, Alexandre Levisse, Marina Zapater, and David Atienza. 2019. BLADE: A bitline accelerator for devices on the edge. In the Great Lakes Symposium on VLSI (GLSVLSI’19). Association for Computing Machinery, New York, NY, 207–212.

[43] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects. In Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'18).

[44] Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li. 2017. FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks. In Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA'17)