gem5-SALAM: A System Architecture for LLVM-based Accelerator Modeling

Samuel Rogers, Joshua Slycord, Mohammadreza Baharani, Hamed Tabkhi
Dept. of Electrical and Computer Engineering
University of North Carolina at Charlotte
Charlotte, North Carolina
{sroger48, jslycord, mbaharan, htabkhiv}@uncc.edu

***Abstract*—With the prevalence of hardware accelerators as an integral part of the modern systems on chip (SoCs), the ability to quickly and accurately model accelerators within the system it operates is critical. This paper presents gem5-SALAM as a novel system architecture for LLVM-based modeling and simulation of custom hardware accelerators integrated into the gem5 framework. gem5-SALAM overcomes the inherent limitations of state-of-the-art trace-based pre-register-transfer level (RTL) simulators by offering a truly "execute-in-execute" LLVM-based model. It enables scalable modeling of multiple dynamically interacting accelerators with full-system simulation support. To create sustainable long-term expansion compatible with the gem5 system framework, gem5-SALAM offers a general-purpose and modular communication interface and memory hierarchy integrated into the gem5 ecosystem which streamlines designing and modeling accelerators for new and emerging applications. Validation on the MachSuite [17] benchmarks present a timing estimation error of less than 1% against Vivado High-Level Synthesis (HLS) tool. Results also show less than a 4% area and power estimation error against Synopsys Design Compiler. Additionally, system validation against implementations on a Ultrascale+ ZCU102 shows an average end-to-end timing error of less than 2%. Lastly, this paper presents the capabilities of gem5-SALAM in cycle-level profiling and full system design space exploration of accelerator-rich systems.**

I. Introduction

With the end of Dennard scaling and the slowdown in Moore's Law, heterogeneous SoCs with many application / domain-specific accelerators have emerged as the major chip design paradigm to deliver never-ending demand for high-performance power-efficient computing. With the high speed at which new algorithms are developed and evolved in comparison to hardware, accelerator designers need rapid prototyping and design space exploration tools. RTL-based simulators, while being very accurate in estimating the timing, power, and area of designs, are often slow and cumbersome to use. As a result, many designers employ partial RTL models [7], [11], [12], [13], [16]. These models generally use software simulators like gem5 [2] for handling system-level modeling, and RTL simulators or RTL-to-C simulators like Verilator [5] for modeling the design elements under testing.

For those who don't want to dive right into RTL development, or want to rapidly prototype and co-design system and data-path elements, there is the option of using pre-RTL modeling and prototyping [4], [18], [19]. These platforms rely on software models for the entirety of simulation, enabling much faster prototyping than RTL design flows. Existing pre-RTL models attain an impressive degree of accuracy, by constraining important system-level details such as control and memory interfaces. Additionally, some of the design decisions are only reasonable for accelerators with uniform operational characteristics that do not change based on data.

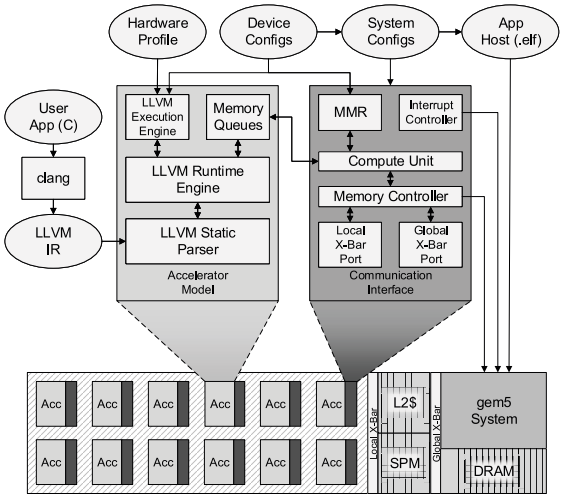


Fig. 1. gem5-SALAM Full-System Architecture

All gem5-SALAM source codes, and procedure to reproduce the results of this research, have been released to the research community, can be found at: **https://github.com/TeCSAR-UNCC/gem5-SALAM.**

In the remainder of this paper, Sec. II outlines the motivation and fundamental need to develop a non-trace based simulation environment. Sec. III lays out the details of our proposed gem5-SALAM simulation model. Sec. IV presents our validation results and co-designing examples for single and multiple accelerators, including concrete examples of various system-level integration of multiple accelerators, which are inherently impossible in trace-based simulators. Sec. V discusses the related work. And Sec. VI concludes this paper.

II. Background and Motivation

One of the biggest challenges for designing and integrating new accelerators into modern SoCs is simulating and exploring design parameters. Often accelerators are initially designed

978-1-7281-7383-2/20/$31.00 ©2020 IEEE
DOI 10.1109/MICRO50266.2020.00047

and tested in isolation, which can lead to over-tuning of design parameters based on idealized assumptions about data availability and other system overheads. While synthesis and RTL simulations of accelerators in isolation may be reasonably quick, full-system RTL simulation of an SoC can often take days or longer. As a result, electronic design automation (EDA) companies, like Synopsys, have developed proprietary tools like Platform Architect that abstract many system-level elements and provide cycle-accurate performance estimations of SoC designs incorporating accelerators through SystemC and transaction-level models (TLM).

For researchers that do not have access to proprietary EDA tools, the gem5 system simulation [2] and its derivatives have become a popular solution. gem5 is an open-source, industry-backed system simulator based on a joint C++/Python programming abstraction. This provides cycle-accurate system performance estimation at an abstraction that is more accessible to developers who are looking for alternatives to proprietary and RTL-based development tools. Furthermore, its open-source C++/Python API enables the extension of its capabilities. While the base API of gem5 supports a wide variety of models for system elements such as memory, CPUs, GPUs, and busses, it lacks modeling for application-specific hardware accelerators. To address this shortcoming, researchers have sought ways to integrate application-specific accelerator modeling into gem5.

gem5-Aladdin [19] integrates the Aladdin [18] pre-RTL simulator into gem5. The base Aladdin simulator relies on instrumenting LLVM IR [9], associated with C descriptions of hardware accelerator functionality, in order to generate a runtime trace of executed LLVM instructions. This runtime trace is then loaded into the simulation engine, where it is further optimized and instrumented with timing information before being passed to an event-driven simulation engine. While Aladdin demonstrates impressive accuracy for a pre-RTL simulation model, its reliance on runtime traces can lead to unreliable results for irregular applications. This is the result of its approach of reverse engineering a datapath based on the parallelism of the dynamic instruction trace. In applications where execution semantics and parallelism vary depending on input data, Aladdin will generate different datapaths for the same kernel source code, as the input data changes. Table I demonstrates this behavior for the Sparse Matrix-Vector Multiplication (SPMV) built around the Compact Row Storage (CRS) data format. In the source code we added a one bit-shift operation that would activate if the input value fell within an arbitrary range we defined, then included this value in one dataset but not the other to demonstrate this shortcomming. As illustrated in Table I, the number of floating-point adders in the datapaths change between two runs with the same kernel code but different input data sets. Additionally, since the value that triggered the shift condition was not in the data set for the first run of the application, Aladdin did not model the shift operation as part of the datapath.

TABLE I
ALADDIN DATAPATH VS DATA-DEPENDENT EXECUTION

Accelerator	Dataset	Functional Units		
		FMUL	FADD	Int Shifter
SPMV-CRS	1	8	4	0
	2	8	8	1

TABLE II
ALADDIN DATAPATH VS. MEMORY DESIGN

Accelerator	Memory		Functional Units	
	Type	Size	FMUL	FADD
GEMM N-Cubed (Fully unrolled)	Cache	256B	665	879
		512B	679	903
		1kB	696	928
		2kB	712	948
		4kB	639	843
		8kB	650	864
		16kB	468	624
	SPM		194	258

The system integration of Aladdin into gem5-Aladdin introduces new limitations. Here, adjustments to system parameters such as cache line size and accelerator cache size, which can impact data availability, have the effect of changing Aladdin's datapath and power estimation for even regular applications. This characteristic is demonstrated in Table II. In this scenario, a sweep of the highly regular GEMM application is run over varying cache sizes. As the sizes of the caches are varied, the number of allocated functional units also changes. Since Aladdin is reverse-engineering the datapath, changes in lookup times and cache hits/misses affect the availability of data and subsequently, the simulated datapath. Table II also shows that switching over to a multi-ported ScratchPad Memory (SPM) has a significant impact on the datapath that Aladdin simulates. While a hardware developer would certainly want to co-optimize both datapath and memory hierarchy, gem5-Aladdin does not provide developers with the means of decoupling the generated datapath from the impacts of the memory hierarchy. Developers should have the capacity to independently sweep design parameters for both datapath and memory[1].

Another limitation of gem5-Aladdin stems from the way it has integrated into gem5's system infrastructure. The gem5-Aladdin project exposes the Aladdin simulator to gem5's system infrastructure by creating a gem5 object wrapper. However, Aladdin's wrapper is only partially integrated into gem5. The gem5-Aladdin's partial integration inhibits system-level exploration and prototyping of common accelerator-rich scenarios in modern SoCs. The gem5 component of gem5-Aladdin merely serves accounting for memory latency in the performance estimates of individual accelerators. As an example, to transfer data between the accelerator's private SPM and dynamic RAM (DRAM), direct memory access (DMA) operations must be exposed directly to the accelerator source code, so that they can appear in the runtime trace. The consequence is that accelerators and their private SPMs lack the standard communication ports necessary for intercommunication between devices in the system. Another example is that MMRs and other traditional

[1]Both the runtime data tests and the memory hierarchy tests were conducted using the latest build of gem5-Aladdin as of April 2020.

interfaces are not included in the accelerator wrapper in gem5-Aladdin. The host CPU instead communicates with accelerators via a software bypass integrated into gem5's Syscall Emulation (SE) simulation framework. Furthermore, these interfaces cannot merely be added to the Aladdin wrapper without violating design assumptions that are imperative to Aladdin's integration into gem5. Doing so would require a complete redesign of the Aladdin wrapper, custom system elements (i.e., DMAs, SPMs, etc.), device drivers, and user programming experience.

Within the scope of pre-RTL and open source tools for system-level modeling of accelerators, there are currently no options that can accurately model runtime-dependent accelerators or the interactions of accelerators with other system elements based on an advanced extensible interfaces (AXI) like communications infrastructure. To address these major limitations, we have identified and incorporated the following contributions into gem5-SALAM:

1) Accurate modeling of datapath structure, area, and static leakage power based on analysis of algorithm-intrinsic characteristics exposed by LLVM.
2) Cycle-accurate modeling of dynamic power and timing through a dynamic LLVM-based runtime execution engine, through gem5-SALAM's dynamic execute-in-execute LLVM-based runtime engine.
3) Separation of datapath and memory infrastructure to enable independent tuning and design space exploration.
4) Flexible system integration that directly exposes accelerator models to other system elements, within gem5, to enable complex inter-accelerator communication and synchronization, using pre-existing gem5 simulation constructs.
5) General purpose C++/Python API for accelerator modeling that decouples computation from system communication, and enables customization and specialization to match user modeling needs.

III. GEM5-SALAM

In this section, we provide an overview of our static front-end setup and initialization, dynamic LLVM runtime engine and metrics evaluation methodology, the API and its integration into gem5, and a description of the simulation setup and configuration.

A. Static Simulator Setup and Initialization

The setup and initializing steps for developing and evaluating an accelerator application within the gem5-SALAM ecosystem utilize the clang compiler tool-chain to format and compile the users application code into LLVM intermediate representation (IR). This IR is statically elaborated by gem5-SALAM's "LLVM Interface" as shown in Fig. 2 to extract the control data-flow graph used for static power and area analysis and by the runtime engine.

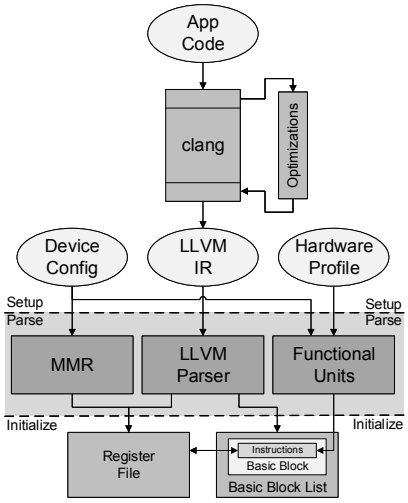


Fig. 2. Accelerator Model Generation

1) IR Generation: To create an application code to run within the gem5-SALAM simulator, the user first writes a functional model of the target hardware accelerator as a single in-lined function in C/C++. For more complex models, each function should be expressed as its own source file. The source code is then compiled into LLVM IR with the use of the clang compiler as shown in Fig. 2. The reason gem5-SALAM defines the granularity of accelerator applications to single in-lined functions is to provide the greatest benefit from clangs optimization passes, such as loop unrolling/vectorization and the removal of internal memory allocation. The resulting structure of the IR allows gem5-SALAM to model power, area, and performance of the accelerator at a high level of accuracy as detailed in Sec. III-C.

Furthermore implementing accelerators at the in-lined function granularity allows for the utilization of clang pragmas and compiler directives for fine-grained control of loop unrolling and vectorization directly in the source code. Additionally this approach aids in the static elaboration of the control and dataflow graph (CDFG) extracted from the IR and the device-specific configuration files described later in Sec. III-E when fed as inputs to the LLVM runtime engine.

2) Static Elaboration: During static elaboration, the accelerator IR and device-specific configuration files are used to extract the static CDFG, and the IR instructions are linked to virtual hardware functional units and registers. The resulting data structure represents a static skeleton of the accelerator data-path, arranged at the granularity of basic blocks as shown in Fig. 3. The Aladdin simulator [18] used a similar approach for functional unit mapping and power modeling that heavily inspired the design of gem5-SALAM's LLVM-based simulator. A major difference is that while Aladdin uses the dynamic CDFG parsed from a runtime execution trace, gem5-SALAM generates its dynamic CDFG at runtime from the static CDFG parsed during static elaboration.

This unique dual CDFG approach enables gem5-SALAM to more accurately model the execution of accelerators with

data-dependent control by independently evaluating static and dynamic elements of the system. This methodology also allows for more configuration knobs for design space exploration. While gem5-SALAM offers a default hardware profile, as shown in Fig. 2, which creates a 1-to-1 map of each instruction to a dedicated functional unit, or the user can define constraints on individual hardware resources to enforce functional unit reuse. gem5-SALAM's ecosystem utilizes additional parameters in the "device config" and "hardware profile" to fine tune the system as detailed later in Sec. III-E.

B. Dynamic LLVM Runtime Engine

The runtime model pictured in Fig. 3 consists of a series of queues controlled by gem5-SALAMs "runtime scheduler" that tracks and evaluates instruction dependencies, allocates hardware resources, and monitors the statuses of in-flight compute and memory operations.

1) Reservation Queue: Execution begins in the "Reservation Queue", pictured on the right in Fig. 3. A dynamic instruction map is generated at the granularity of basic blocks from the statically elaborated CDFG, and the contents of the first basic block of the application are imported. As each instruction (operation) is added to the queue, dynamic dependencies are generated by searching upward in the reservation queue as well as the in-flight compute and memory queues. Additionally, the execution of previous instances of the same instruction and all instructions that read from its destination register are checked to be in-flight or completed. This ensures each instruction can only be launched once all of its dependencies have been met.

Instructions that function as basic block terminators trigger the reservation queue to load the next basic block immediately after evaluation. This enables the simulation of a custom, highly-parallelizable, data-path with support for the pipelining of loop structures. Compute instructions, which have an associated hardware unit mapping, have the additional constraint that their mapped hardware unit is available if resource limitations are defined by users. This enables the user to enforce reuse in portions of the data-path via the "Hardware Profile" provided in the front-end, as shown in Fig. 2. When an instruction is ready to execute, it is then transferred to an appropriate operation queue.

2) Compute Queue: Compute instructions are all instructions that can be resolved by the simulator using only values stored in local registers. These instructions are transferred to the "Compute Queue" where their functional units are invoked. For simulation purposes, the computation is done immediately, but the commit of the result can be delayed by a number of operational cycles that is uniquely configurable for each function unit type. The dynamic energy of each active instruction is also calculated at this point to estimate the power of the compute data-path. Once an instruction (operation) is ready for commit, the instruction is removed from the queue, the hardware unit is released, and the Reservation Queue is signaled to resolve dependencies on the committed instruction.

3) Memory Queues: Memory instructions will be transferred to the Read/Write queues shown in the bottom-right corner of Fig. 3. These queues forward the memory requests to the connected communications interface, described in Sec. III-D, which is responsible for interfacing with gem5's other system elements. The memory queues operate asynchronously from other elements of the runtime engine in order to handle memory requests that complete in between the compute cycles of the runtime engine. When a memory request is ready to commit, the request is removed from the queue and the "Reservation Queue" is signaled to resolve dependencies on the committed request.

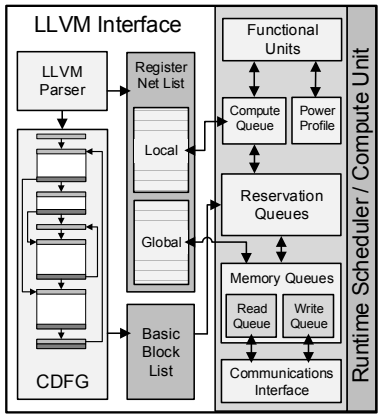


Fig. 3. LLVM Runtime Engine Simulation Model

C. Metrics Estimation

All of the elements that make up the simulator perform some form of internal statistics tracking that is fed into the "LLVM Interface" during each phases of operation. The statically elaborated CDFG provides the baseline model for static power and area, while the dynamic runtime generates and records evaluation data each cycle during simulation. Because there are also so many configurable knobs, a brief overview of the related parameters will be in each subsection below, with greater detail in Sec. III-E.

1) Power and Area: The power estimation model utilizes parameters defined within the hardware profile and the device config as shown in Fig. 1 and Fig. 2. The hardware profile contains power and area profiles for common fixed and floating-point hardware functional units as well as single bit registers operating with various latency's. The generation of this profile is detailed below in Sec. IV-A. The device config allows the user to constrain the amount of each hardware functional unit that is in the system. The static power metrics use the static CDFG to account for all functional units within the system, the simulation runtime, and the hardware profile to determine the leakage power lost in the system due to the functional units. The dynamic power used by the functional units is calculated each cycle for each active functional unit and is the combination of the switching and internal power dissipation as defined in the hardware profile as a function of the accelerator clock speed, which is defined in the device config.

Similarly the LLVM IR as used in gem5-SALAM exposes the internal registers and their bit size, while the runtime engine

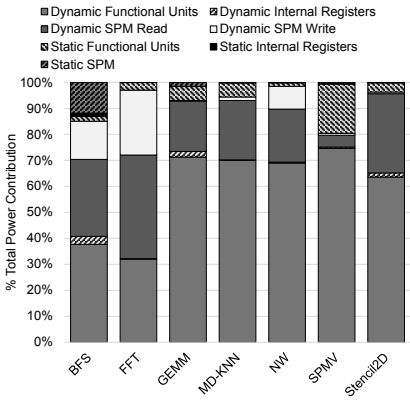


Fig. 4. Example of total power analysis of multiple benchmarks using private SPM.

tracks the read and write activity each cycle. This allows gem5-SALAM to also model the runtime energy consumption of internal data-path logic using the same method as described for the functional units, where the static and dynamic power and area are calculated based around the single-bit register results obtained for the hardware profile.

By utilizing gem5's memory interface, gem5-SALAM also has built-in support for power modeling of the shared memory with respects to user-defined configurations to the gem5 system. To account for situations where the user prefers private memory elements integrated within an individual accelerator, gem5-SALAM takes advantage of McPat's Cacti [10] by automatically passing private memory parameters and usage statistics internally to provide the power and area profile upon runtime completion. Fig. 4 shows the type of results generated when performing full power analysis for multiple MachSuite [17] benchmarks ran in parallel with private SPM.

2) Performance and Occupancy: gem5-SALAM also provides a variety of performance metrics to the user post-simulation. Within the device configuration, gem5-SALAM defines the cycle time that each LLVM IR instruction takes to execute in the compute queues, where the default values were tuned and validated vs HLS performance below in Sec. IV-A. The user can define the latency of hardware devices and the clock-speed within the accelerator. These knobs enable users to accurately model and explore their effects on cycle-counts, runtime, and functional unit occupancy of accelerator models.

During the dynamic runtime simulation gem5-SALAM logs which instructions are scheduled or in-flight for each cycle. This additional data point combined with configurable hardware resources allows for a fine grained analysis and exploration tool for exploring occupancy levels within the system. Some examples of this that are explored more in Sec. IV include the ability to view functional unit occupancy as a function of data-availability by sweeping port sizes or optimizing functional unit resources for maximum parallelism.

D. gem5 Integration and Scalable Full System Simulation

gem5 provides a robust, extensible, and well-tested framework that makes it ideal for evaluating new heterogeneous architectures. Unlike other simulators that built another simulator before integrating into gem5, gem5-SALAM was built from the ground up within the modular APIs offered by gem5. **gem5-SALAM does not require a rebuild of gem5 to add new accelerators.**

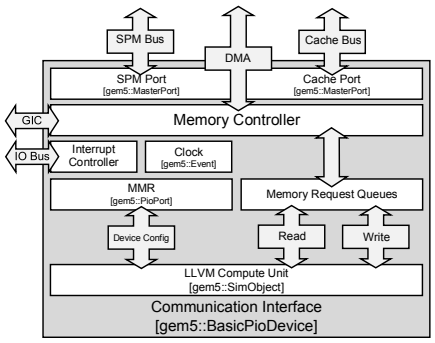


Fig. 5. Communications Interface

Since accelerator models are built on top of native gem5 constructs, they can be integrated anywhere within a gem5 simulation instance that supports a "gem5::TimedObject". This gives designers the freedom to explore both tightly and loosely coupled accelerator designs and even nest accelerators within the datapaths of other system elements. Additionally, gem5-SALAM offers multiple types of DMA devices including block and stream DMAs. gem5-SALAM is the first and only gem5 extension to provide a full suite of extensible simulation models for pre-RTL and pre-HDL design space exploration of application-specific hardware accelerators.

1) Compute Unit and Communications Interface: At the core of our API are two basic models: the Compute Unit and the Communications Interface. A compute unit represents the datapath of a hardware accelerator. An example of this is described in Sec. III-B, however that is the API provided by gem5-SALAM allows for the construction of other simulation models that can hook cleanly into the rest of gem5-SALAM's system infrastructure.

A Communications Interface, shown in Fig. 5, provides access to the system interfaces of gem5 for the purposes of memory access, control, and synchronization. It accomplishes this by providing three basic interfaces in its API: Memory-Mapped Registers (MMRs), memory master ports, and interrupt lines. Fig. 5 shows the the most basic model of a "Communications Interface", or the "CommInterface". It supports programming via its MMR and access to memory through up to two master memory ports. This enables designers to generate accelerators with parallel access to different memory types in parallel, including SPMs and caches. Furthermore, gem5-SALAM supports more specialized memory access types, such as stream buffers and SPMs with customized partitioning. To demonstrate the extensibility of the gem5-SALAM API, custom interfaces supporting stream inputs and custom-ported memories have been built upon the base "CommInterface" model, that integrate seamlessly with the LLVM Runtime engine, and are employed in the architecture explorations described in Sec. IV.

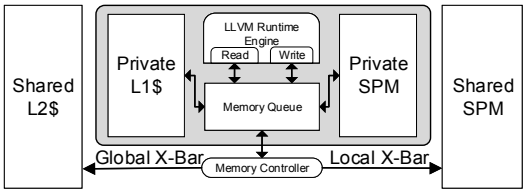


Fig. 6. Accelerator Memory Model

A user-configurable memory controller enables the distribution of parallel memory access across all memory interfaces as shown in Fig. 6. For private memory and streaming interfaces, the memory controller also supports the configuration of memory partitioning and bandwidth. Read and write request queues allow tracking of in-flight memory requests, and will automatically notify the Compute Unit when memory requests have been fulfilled. Additionally, the clocks of the Communications Interface and Compute Unit can be configured independently.

Empowering users to explore innovative accelerator designs and hierarchies was a major design goal with gem5-SALAM. As a result, all of the Communication Interfaces are interchangeable, without requiring any modification of the corresponding Compute Unit. This stands in contrast to other simulators, like gem5-Aladdin and PARADE, that are unable to decouple the execution models of their accelerators from their control and communications interfaces.

2) Multi-ACC Simulation: gem5-SALAM was built with multi-accelerator designs in mind. The configurable Communications Interfaces enable communication and the sharing of data between hardware accelerators. In order to introduce some degree of device hierarchy, and simplify configuration from the user perspective, gem5-SALAM provides a hierarchical Accelerator Cluster construct. An accelerator cluster consists of a pool of accelerators coupled with a shared DMA and scratchpad. A local crossbar provides access to shared resources as well as the MMRs of other accelerators in the cluster. This enables accelerators to communicate directly with each other and access shared data. Accelerators within a cluster can still be configured with private scratchpads and other memory interfaces. A global crossbar is also included to grant access to resources outside of the cluster, such as DRAM. If caches are enabled, a last-level cache is added between the global crossbar and system memory interface to enable cache coherency between accelerator clusters and other processing elements.

This setup enables numerous opportunities for design space exploration of accelerator rich systems. For one, users have the ability to track memory statistics such as bandwidth utilization and cache misses on shared system resources. Alternatively, accelerator clusters can be used to construct templates for complex accelerator tasks that can be replicated for parallel execution. Importantly, the capability for accelerators to communicate and self-synchronize in gem5-SALAM reduces host CPU overheads for control and synchronization, This enables the system and simulation to scale better with a larger number of accelerators than other pre-RTL simulators.

3) Control and Synchronization: Control of accelerators within gem5-SALAM is largely enabled via memory-mapped registers. Each of the Communications Interfaces described above comes equipped with configurable status, control, and data registers. This enables low-level device configuration as well as basic synchronization controls. Used in conjunction with the other memory interfaces, this enables direct communication and coordination between and accelerator and host processor, and even between accelerators. Additionally, each Communications Interface also supports the generation of interrupts to the system interrupt controller.

For the synchronization, by default, our accelerator models are capable of generating interrupts through the ARM GIC. Additionally, the MMRs of accelerators are set to respond with their current values when read by the host CPU. The CPU's perspective of the accelerator is the same as it is for any other memory-mapped device. The accelerated portion of the host code is replaced with a device driver that sets the accelerator MMRs and performs and necessary data movement between host and accelerator memories. Drivers for DMAs are included in our project files. Drivers for accelerators are highly device-specific, but templates are provided to simplify the development process.

gem5-SALAM utilizes gem5's Full System simulation mode, as opposed to Syscall Emulation. To simplify driver development, the simulation is run with a bare-metal kernel. gem5-SALAM also supports simulation with a full Linux kernel (provided by gem5), however drivers will need to be adapted to map virtual memory addresses of device MMRs.

E. Simulation Setup and Configuration

Setting up a new simulation in gem5-SALAM has been streamlined as much as possible to require minimal effort from the end-user and to be language agnostic with the use of LLVM. The simulation profile needed to run the simulation can be divided into two main categories: (1) single accelerator configuration and (2) accelerators cluster configuration.

1) Single Accelerator Configuration: Each accelerator must first be configured independently before being added to the accelerator cluster. Each accelerator configuration contains the accelerated code segment, which is passed through the Clang compiler to generate the LLVM IR used by the simulator as in Fig. 7. The user can customize the underlying structure of the datapath by applying compile-time optimizations like loop-unrolling and vectorization at this point in time. Within the host code itself, the user must define the locations of Memory-Mapped Registers (MMRs) to be used by the accelerator. Similar to the programming abstraction of OpenCl or CUDA, the inputs and outputs are exposed as pointers within the accelerated function declaration. The user must then map these pointers to the MMRs of the accelerator, along with any other configuration variables/flags. This means that the programmer can change where an accelerator reads/writes its data at runtime through a device driver. Overall, minor changes need to be made to the application's host code to map the memory to the

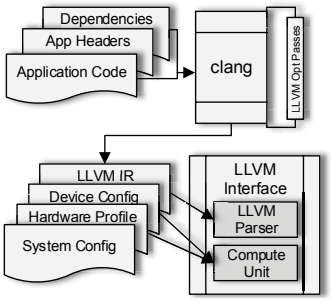


Fig. 7. Single Accelerator Configuration

device. For example, if the use of DMA transfer is desired by the user then "memcpy" needs to be replaced by "dmacpy" in the application's host code.

Alongside the application's host and device codes, gem5-SALAM requires gem5-python device and system configuration files and the hardware profile. The system configuration file sets the gem5 specific interface parameters including: the number and size of ports, MMR base addresses and sizes, and accelerator memory ranges used to route data within the system. The device configuration is used to customize the configuration of the accelerator datapath and tune runtime parameters based on profiling data provided during simulation. This file includes options for customizing memory interfaces, device clocks, and setting datapath constraints. These configurations are passed to gem5-SALAM and the internal communications interface to define the interconnect between accelerator model and other simulation components in gem5. Additionally, within the device configuration there are a few options for configuring the dynamic runtime scheduler. Examples of each type of configuration file are provided to guide users in the design of accelerators. Alternatively users can control and sweep the same design parameters directly in gem5's Python API. This can be useful if using Python for design space sweeps, or integrating with other projects based on gem5.

2) Accelerators Cluster Configuration: One significant feature of gem5-SALAM is empowering users to construct rapid simulation models of accelerator clusters. The accelerator cluster can contain any number of accelerators and the shared resources defined by the user between them. The cluster is generated through gem5-python scripts to initialize each of the individual accelerator and system elements as defined by the gem5-python device and system configurations shown in Fig. 8. The hardware accelerator cluster configuration automates the interconnection and initialization of accelerators. To facilitate

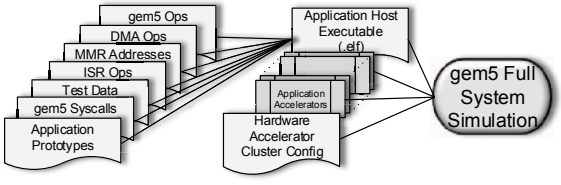


Fig. 8. Accelerator Cluster Configuration

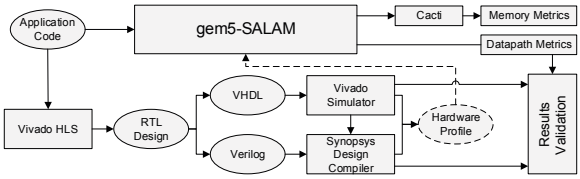


Fig. 9. Validation Flow

this, gem5-SALAM provides a library of python classes that represent the hardware components with a C++ wrapper that passed arguments directly into our simulator and allows the user to reconfigure the device and system files without the need to recompile the base code.

IV. Simulation Results and Validation

To demonstrate the benefits of gem5-SALAM, we have classified the results into three categories: (1) Metric validation, (2) single accelerator design space exploration, (3) multiple accelerators design space exploration. In the following, we present the results and analysis in detail.

A. Metrics Validation

Fig. 9 presents the validation flow for timing, power and area validation. The timing model of gem5-SALAM was validated on the MachSuite [17] benchmarks against RTL models generated by Vivado HLS. The power and area models for functional units in gem5-SALAM are based on the models used in gem5-Aladdin [19]. These models were also validated on the MachSuite [17] benchmarks against Synopsys Design Compiler, using an open source 40nm standard cell library and the gate switching activity produced by RTL simulation in Vivado. This validated hardware profile is included as the default configuration in gem5-SALAM, although the user can easily modify or extend this profile to explore custom hardware.

Fig. 10 shows the timing performance validation for 8 benchmarks from MachSuite. Overall, the average timing error was approximately 1%. In each case, the input LLVM IR was tuned to reflect the same levels of Instruction Level Parallelism (ILP) as the datapaths generated by HLS. Applications like FFT (0.32% error), GEMM (0.32% error), and Stencil2D (0.13% error) had some of the lowest timing errors due to their highly regular, data-independent control. NW also exhibited a very low timing error of 0.19% due to the mapping of many of its runtime control dependencies to MUXs in both HLS and SALAM. The highest error appears in MD-KNN which relies very heavily on floating-point computation. HLS tools will generally attempt to minimize the number of floating-point

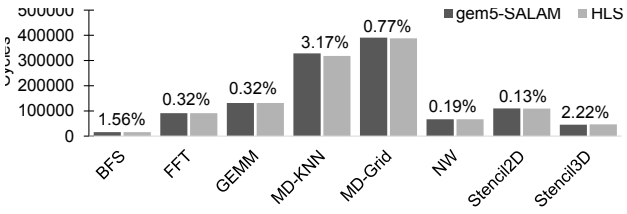


Fig. 10. Performance Validation

TABLE III
SYSTEM VALIDATION RESULTS

Benchmarks	FPGA			Simulation			Error (%)		
	Compute Time (uS)	Bulk Xfer Time (uS)	Total Time (uS)	Compute Time (uS)	Bulk Xfer Time (uS)	Total Time (uS)	Compute Time	Bulk Xfer Time	Total Time
FFT/Strided	879.35	93.58	972.93	867.77	95.58	963.35	1.32	-2.14	0.98
GEMM/ncubed	1343.31	179.01	1522.32	1315.24	181.97	1497.21	2.09	-1.65	1.65
Stencil2D	846.45	268.57	1115.02	854.14	275.98	1130.12	-0.91	-2.76	-1.35
Stencil3D	445.28	444.5	889.78	455.26	446	901.26	-2.24	-0.34	-1.29
MD/KNN	2489.66	118.74	2608.4	2568.45	112.96	2681.41	-3.16	4.87	-2.80
Average	-	-	-	-	-	-	1.94	2.35	1.62

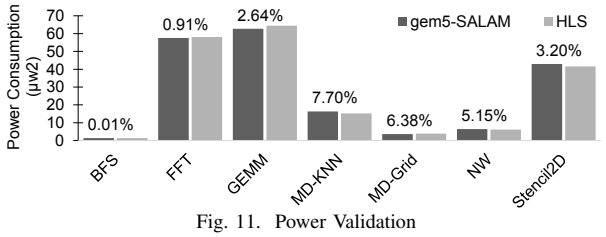


Fig. 11. Power Validation

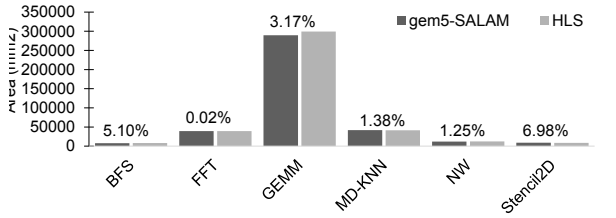


Fig. 12. Area Validation

functional units employed in a design, and enforce reused of expensive floating point resources. We validated gem5-SALAM by enforcing similar restrictions and reuse in the configuration of the MD-KNN accelerator, however the runtime mechanism for functional units employed by gem5-SALAM only approximates the internal wiring of those reuse circuits. Even so, the power and area estimates detailed below for the same accelerator justify gem5-SALAM's means of modeling functional unit reuse by showing low levels of error in both power and area.

Fig. 11 shows the power validation across the same set of benchmarks. Stencil3D was excluded from this set due to Design Compiler running out of memory during elaboration. The average error in power estimation is slightly higher, at 3.25%. The MD-KNN, MD-Grid, and NW benchmarks show the highest power error due to heavier reliance on muxes and non-arithmetic operators. Variability in the power consumption of these operators leads to a slight overestimation of power requirements on average. These results are very comparable to those produced by Aladdin.

Fig. 12 shows the area validation on the evaluated benchmarks. On average gem5-SALAM is able to estimate chip area with an error of 2.24%. MD-Grid was excluded from this test due to custom IPs within its data-path preventing Design Compiler from providing area estimations.

B. System Validation on FPGAs

For the purpose of system validation, we synthesized five of the benchmarks and executed them on a Xilinx Zynq UltraScale+ MPSoC ZCU102 evaluation board, which has the XCZU9EG SoC chip. The ARM processors clocked at 1.2GHz. We used Vivado HLS 2018.3 to synthesize the benchmarks and Vivado SDSoC 2018.3 to cross-compile the host programs, which invoke the kernel synthesized by Vivado. The targeted benchmarks are summarized in Table III. The reported bulk transfer time is the summation of both read/write time from/to shared DDR memory. To match the configuration of the FPGA programmable logic a accelerator cluster was instantiated within gem5-SALAM consisting of a DMA, an accelerator for the top-level function, and an accelerator for the benchmark kernel. The top accelerator was programmed by the host CPU and used to schedule memory transfers and invoke the benchmark accelerator. The burst width of the cluster DMA was tuned to match the burst width of the data mover.

Table III displays a similar trend to that seen in the comparison to RTL simulation. Positive error indicates when simulation was faster, while negative errors indicated faster FPGA times. The biggest discrepancies vs. the previous comparison can be seen in GEMM and FFT. These two benchmarks operate on double-precision floating point, whereas most of the other benchmarks operate on integer types. By default, gem5-SALAM approximates floating point operations using 3-stage FP adders and multipliers, which do not precisely match the floating point DSP IPs employed by SDSoC. Even so, the timing is close enough to maintain a high degree of fidelity with the FPGA implementation. On average the absolute compute error across all benchmarks was 1.94%. Likewise, the average absolute error in data transfer times was 2.35%. This is primarily due to a difference in cache invalidation times between the ZCU102 and the simulation.

C. Simulation Time vs. gem5-Aladdin

We compared the preprocessing and simulation times of gem5-Aladdin and gem5-SALAM using a system with an i7-7700 and 16GB of RAM. Table IV shows the results across 9 Machsuite benchmarks. While preprocessing in gem5-Aladdin require binary instrumentation and runtime trace generation, the only preprocessing required by gem5-SALAM is the compilation of the accelerated kernel. This results in an average preprocessing speedup of 123x vs. gem5-Aladdin.

Simulation overheads in gem5-Aladdin are also much higher, requiring the loading of large trace files to memory, trace graph optimization, and finally graph execution. By comparison the memory footprint of gem5-SALAM is much smaller, operating on the static CDFG and maintaining small runtime queues to

TABLE IV
SIMULATOR SETUP AND RUNTIME EXECUTION TIMING.

Benchmark	gem5-Aladdin		gem5-SALAM		Speedup (SALAM v. Aladdin)	
	Trace-Gen	Simulation	Compilation	Simulation	Preprocess	Simulation
BFS	1.4E-1s	13.6s	2.5E-4s	5.0E-2s	555x	273x
FFT	5.8E-1s	37.1s	2.2E-2s	9.5s	26x	3.9x
GEMM	9.0s	130.1s	5.6E-2s	19.7s	160x	6.5x
MD-Grid	2.5s	36.3s	1.2E-2s	2.8E-1s	211x	132x
MD-KNN	5.4E-1s	45.5s	4.8E-2s	2.5E-1s	11.3x	180x
NW	7.0E-1s	11.1s	1.5E-2s	5.5E-1	47x	20x
SPMV	1.0E-1s	50.8s	1.8E-2s	4.6E-2	5.9x	1113x
Stencil2D	2.3s	55.2s	3.5E-1s	1.3s	6.6x	43.8x
Stencil3D	1.5s	86.6s	1.8E-2s	1.9E-2s	81.5x	4503x
Average	-	-	-	-	123x	697x

hold the dynamic operation context. This results in an average simulation time speedup of 697x vs. gem5-Aladdin, with a maximum observed speedup of over 4503x on the Stencil3D application.

D. Design Space Exploration

1) Case Study: Generic Matrix Multiply: To show the capabilities of gem5-SALAM we have provided an example of the design space exploration that can be achieved by looking at the General Matrix multiply (GEMM) application. A simple bash script was created to sweep the quantity of available functional units defined in the device configuration shown in Fig. 7 for a range of memory bandwidth allocations as defined in the accelerator cluster configuration shown in Fig. 8 to determine the benefits of memory parallelism with the GEMM application. The output of each of these simulations was exported in CSV format and combined to enable analysis of the power and performance estimations as a Pareto curve in Fig. 13. Here, there is an interesting trend of duplicate results with higher power consumption that suggests over-allocation of functional units versus the runtime parallelism that exists in the accelerator. One source of this discrepancy arises from limitations in memory bandwidth.

Fig. 14(a) presents how the proportion of stall cycles to running cycles improves as we increase the memory bandwidth. Supporting more than 64 read/write ports in the design provides no additional benefit since this is the maximum width of the data-path. Observing that the amount of stalled cycles is still higher than cycles that scheduled new instructions, we can break the stall sources down even further as shown in Fig. 14(b). This graph allows us to see that the design space for GEMM is most heavily influenced by floating-point computations and data

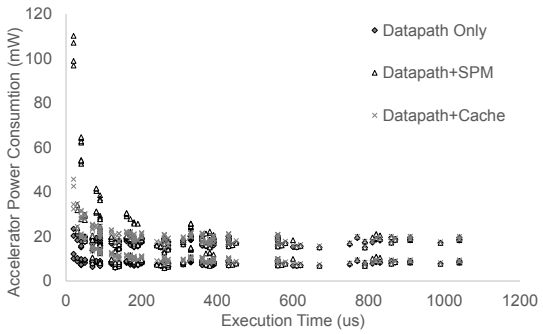
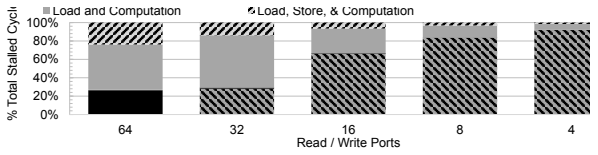


Fig. 13. GEMM Design Space Pareto Curve

(a) Runtime instruction scheduling comparison



(b) Runtime stall cycle unifinished operation breakdown

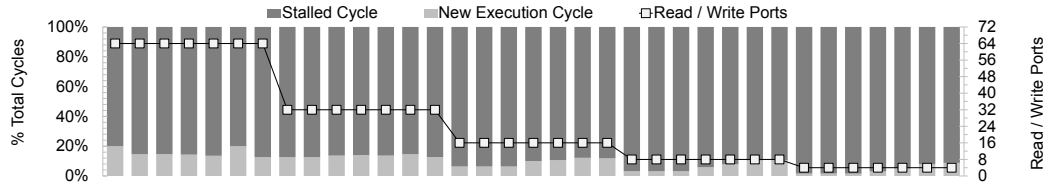
Fig. 14. GEMM Stalls Breakdown

transfer into the accelerator. The solid black bands representing cycles of only floating-point computation. In the 32 and 64 port columns, Fig. 14(b) indicates that by increasing the bandwidth we are also creating more temporal parallelism in the nested floating-point operations that are decoupled from memory operations.

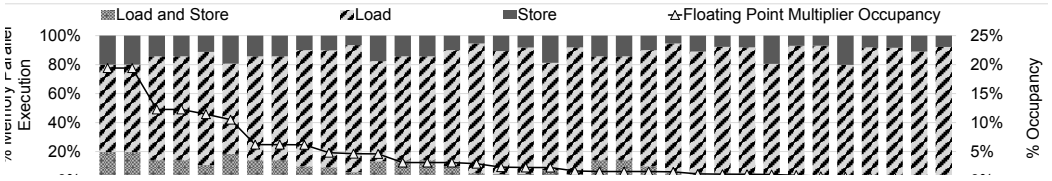
2) Co-Designing using gem5-SALAM: By using gem5-SALAM to examine functional unit occupancy at a cycle granularity, we find low occupancy among floating point adders. Additionally we can determine that 64 total floating points addition units in our accelerator can provide nearly the same throughput as 128 with only an increased performance cost of 4 cycles. Using this now as our basis we will hold the number of floating-point addition units at 64 and re-evaluate our design space domain.

Fig. 15 shows a sampling of the additional exploration pathways available to the user with the use of gem5-SALAM to aid in the co-design of accelerator applications. By first using average values across a wide range of sweeps, we have quickly and effectively narrowed the scope of the design space such that we can now explore each remaining path directly. In Fig. 15(a), we repeated our initial experiment for each configuration individually and plotted the stalled cycles versus cycles where new operations where executed for the remaining sweeps. We can now explore the parallelism between memory operations and floating-point operations. The details are provided in Fig. 15(b) which examines the cycle execution scheduling activities rather than the stalls as in Fig. 14(b). Fig. 15(b) shows the overlap between load and store operations within the application and overlay the average occupancy of the floating-point multiplication units. These two elements for each column show a clear trend of higher occupancy levels for sweeps with minimal overlap between load and store operations.

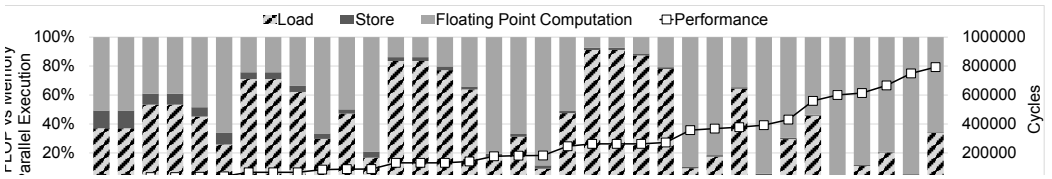
To further explore the analysis, Fig. 15(c) incorporates the floating-point computation scheduling activities into the results and overlays the overall performance for each sweep. This allows us a new insight into the causation of the previous results. We can observe that the optimal performance is obtained when the ratio of operations scheduled is nearly the same as the ratio of floating-point operations to memory operations in the GEMM algorithm. Looking at these same metrics from



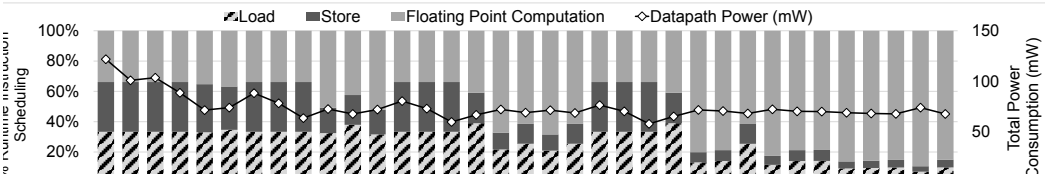
(a) Exploration of Datapath Stalls vs Memory Ports



(b) Memory Parallelism vs Floating Point Function Unit Occupancy



(c) Memory-to-Compute Ratio vs Execution Time



(d) Memory-to-Compute Ratio vs Power Consumption

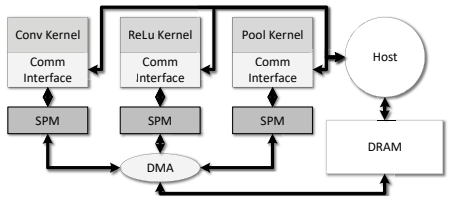
Fig. 15. GEMM Memory and Compute Design Space Exploration

a different perspective, Fig. 15(d) evaluates which type of instruction on average is scheduled each cycle with an overlay showing the total power consumption.

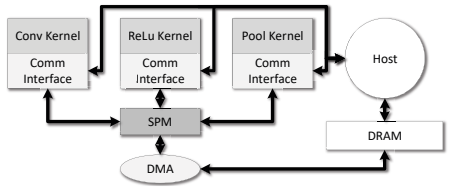
E. Multi Accelerator Design Space Exploration

One of the key benefits of gem5-SALAM over other existing pre-RTL simulators is its increased support and flexibility for design space exploration of multi-accelerator workloads. Flexibility in system interconnects and hierarchical models like the accelerator cluster enable simulation of complex hardware accelerator interactions not available in gem5-Aladdin or PARADE.

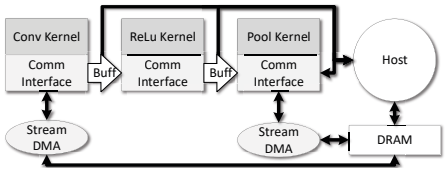
To demonstrate this, we implemented the first layer of a Convolutional Neural Network (CNN) in gem5-SALAM. This consisted of dedicated accelerators for the 2D convolution, max pooling, and rectify linear (ReLU) functions. The cluster of accelerators was evaluated in three different scenarios. In the first scenario, shown in Fig. 16(a), each accelerator used its own private memory. Similar to the semantic supported by gem5-Aladdin, DMAs were responsible for data movement between accelerators and the host was responsible for activating and synchronizing the accelerators. For the purposes of timing comparison, this scenario serves as the baseline for comparison. In the second scenario, shown in Fig. 16(b), accelerators have a shared scratchpad memory for directly passing data to each other, but no way of knowing when their data is available. In this scenario synchronization with the a central controller is necessary to maintain synchronization across accelerators, similar to the model in PARADE. The removal of data movement between accelerators results in a 25% speedup in the end-to-end execution, but the requirement for external synchronization still limits overall performance. In the third scenario, shown in Fig. 16(c), accelerators communicate directly with each other through stream buffers that function in a similar fashion to the AXI-Stream interfaces employed in modern ARM-based SoCs. In this scenario no centralized controller is needed or used to synchronize the operation of the accelerators. By enabling inter-accelerator pipelining, the end-to-end execution is improved by a factor of 2.08x over the baseline. gem5-SALAM, which simulates all three scenarios, is the only simulator of the three that is capable of modeling this multi-accelerator integration. This is due to the fact that this style of streaming data transfer requires a two-way handshake for synchronizing data movement between devices that may internally operate with different data rates,



(a) Private Scratchpad Memory



(b) Shared Scratchpad Memory



(c) Direct Communication

Fig. 16. Producer-Consumer Accelerator Scenarios

or otherwise experience fluctuations in their data rates due to runtime variables. The black box accelerator models of gem5-Aladdin and PARADE lack the fundamental interfaces required to facilitate this form of communication. More importantly, the inability to decouple basic control and communication interfaces from the execution models of gem5-Aladdin and PARADE without significant redesigns to their underlying structures means that they will continue fall even further behind as the complexity of SoC designs increase. gem5-SALAM's API was built to be easily extensible and inherently support integration with the work of other developers in the gem5 ecosystem.

V. Related Work

As hardware design has shifted from classical RTL design flows to more software developer-friendly approaches the LLVM compiler [9], and its IR, has become a key component of many design flows. Popular HLS tools like Vivado and LegUp [3] internally use a modified clang tool flow for translating hard descriptions written in C to popular RTL targets such as Verilog, VHDL, and SystemC. With growing interests in deep learning acceleration the LeFlow project [14] was developed to integrate TensorFlow's XLA compiler with LegUp and enable the synthesis of deep learning accelerators on FPGAs. In addition to synthesis tools, LLVM has also been employed for pre-RTL design space exploration. The RIP framework [21] leverages LLVM for the identification and modeling of "hot loops" in an application in order to design accelerators for those portions of code. Similarly, Needle [8] and the work described in [15] leverage LLVM for detecting hot portions of code and automatically generating accelerators for DySER-styled [6] architectures. For exploring the design and system-level impacts of loosely-coupled hardware accelerators Lumos+ [20] and LogCa [1] tools can be used. These approaches employ analytical modeling for estimating the power, performance, and area requirements of hardware accelerators in highly heterogeneous accelerator-rich systems. More recently the MosaicSim tool (ISPASS 2020), which relies on LLVM instrumentation and parsing for modeling accelerators, was released to offer lightweight simulation of heterogeneous systems comprised of CPUs and accelerators. Much like gem5-Aladdin it relies on binary instrumentation and trace-based simulation to model the runtime characteristics of hardware accelerators. MosaicSim employs a simplified simulation framework that abstractly models various CPU designs as well as accelerators, making it significantly faster to simulate designs than many other simulators, including gem5 and its derivatives. This comes at the cost of simulation fidelity, power and area modeling, the integration of GPUs, and the usage of SystemC-based design flows supported by other simulators like gem5.

As mentioned in Sec. II, other existing pre-RTL solutions for exploring the system-level integration of accelerators are gem5 [2] and its derivatives gem5-Aladdin [19] and PARADE [4]. While gem5 offers a high degree of flexibility in system-level design space exploration, it lacks any base models for integrating application-specific hardware accelerators. gem5-Aladdin and PARADE offer such modeling capabilities, but do so by heavily constraining the design space to align with their particular simulation semantics. Furthermore, the accuracy of their modeling is limited to the scope of accelerators in which data availability, compute parallelism, and timing are independent of the input data and system hierarchy.

For researchers who are more comfortable with SystemC development, gem5 now supports the direct integration of SystemC models [12]. This offers the most opportunities for design space exploration and simulation, however, as an RTL-based option, it will also require a higher degree of design effort than the other options.

VI. Conclusions

This paper presented gem5-SALAM, as a fully integrative LLVM-based simulation platform for salable simulation of accelerator-rich SoCs. Unlike the existing simulation platform, gem5-SALAM offers the run-time engine as a new Sim-Object within gem5 ecosystem to create a flexible full system simulation with many hardware accelerators. It takes unmodified LLVM code generated from any language, as well as the desired accelerators and system configurations, and automatically creates a full system simulation within the gem5 ecosystem. Validation results demonstrated the performance estimation error of less than 2% and area and power estimation errors of less than 4%. The paper also presents significant benefits of gem5-SALAM in enabling full system simulations and design space exploration for single and multiple accelerators with varying design sweeps.

REFERENCES

[1] M. S. B. Altaf and D. A. Wood, "Logca: A high-level performance model for hardware accelerators," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, June 2017, pp. 375–388.

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[3] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, and J. H. Anderson, "Legup: An open-source high-level synthesis tool for fpga-based processor/accelerator systems," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 2, pp. 24:1–24:27, Sep. 2013. [Online]. Available: http://doi.acm.org/10.1145/2514740

[4] J. Cong, Z. Fang, M. Gill, and G. Reinman, "Parade: A cycle-accurate full-system simulation platform for accelerator-rich architectural design and exploration," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015.

[5] K. Gent and M. S. Hsiao, "Functional test generation at the rtl using swarm intelligence and bounded model checking," in *2013 22nd Asian Test Symposium*, Nov 2013, pp. 233–238.

[6] V. Govindaraju, C.-H. Ho, and K. Sankaralingam, "Dynamically Specialized Datapaths for energy efficient computing," in *High Performance Computer Architecture (HPCA)*, 2011, pp. 503–514.

[7] K. Iordanou, O. Palomar, J. Mawer, C. Gorgovan, A. Nisbet, and M. Luján, "Simacc: A configurable cycle-accurate simulator for customized accelerators on cpu-fpgas socs," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 2019, pp. 163–171.

[8] S. Kumar, N. Sumner, V. Srinivasan, S. Margerm, and A. Shriraman, "Needle: Leveraging program analysis to analyze and extract accelerators from whole programs," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017, pp. 565–576.

[9] C. Lattner and V. Adve, "Llvm: a compilation framework for lifelong program analysis transformation," in *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, 2004, pp. 75–86.

[10] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42Nd Annual IEEE/ACM*

[18] Y. S. Shao, B. Reagan, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014.

International Symposium on Microarchitecture, ser. MICRO 42. New York, NY, USA: ACM, 2009, pp. 469–480. [Online]. Available: http://doi.acm.org/10.1145/1669112.1669172

[11] T. Liang, L. Feng, S. Sinha, and W. Zhang, "Paas: A system level simulator for heterogeneous computing architectures," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2017, pp. 1–8.

[12] C. Menard, J. Castrillon, M. Jung, and N. Wehn, "System simulation with gem5 and systemc: The keystone for full interoperability," in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, July 2017, pp. 62–69.

[13] T. Nikolaos, K. Georgopoulos, and Y. Papaefstathiou, "A novel way to efficiently simulate complex full systems incorporating hardware accelerators," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 658–661.

[14] D. H. Noronha, B. Salehpour, and S. J. E. Wilton, "Leflow: Enabling flexible FPGA high-level synthesis of tensorflow deep neural networks," *CoRR*, vol. abs/1807.05317, 2018. [Online]. Available: http://arxiv.org/abs/1807.05317

[15] T. Nowatzki and K. Sankaralingam, "Analyzing behavior specialized acceleration," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 2. ACM, 2016, pp. 697–711.

[16] C. Pham-Quoc, I. Ashraf, Z. Al-Ars, and K. Bertels, "Heterogeneous hardware accelerators with hybrid interconnect: An automated design approach," in *2015 International Conference on Advanced Computing and Applications (ACOMP)*, Nov 2015, pp. 59–66.

[17] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, "MachSuite: Benchmarks for accelerator design and customized architectures," in *Proceedings of the IEEE International Symposium on Workload Characterization*, Raleigh, North Carolina, October 2014.

[19] Y. S. Shao, S. Xi, V. Srinivasan, G.-Y. Wei, and D. Brooks, "Co-Designing Accelerators and SoC Interfaces using gem5-Aladdin," in *The 49th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[20] L. Wang and K. Skadron, "Lumos+: Rapid, pre-rtl design space exploration on accelerator-rich heterogeneous architectures with reconfigurable logic," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, Oct 2016, pp. 328–335.

[21] W. Zuo, L. Pouchet, A. Ayupov, T. Kim, Chung-Wei Lin, S. Shiraishi, and D. Chen, "Accurate high-level modeling and automated hardware/software co-design for effective soc design space exploration," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2017, pp. 1–6.