gem5-accel: A Pre-RTL Simulation Toolchain for Accelerator Architecture Validation

João Vieira*, Nuno Roma*, Gabriel Falcao†, Pedro Tomás*

*INESC-ID, Instituto Superior Técnico, University of Lisbon, Portugal
†Instituto de Telecomunicações, University of Coimbra, Portugal

***Abstract*—Attaining the performance and efficiency levels required by modern applications often requires the use of application-specific accelerators. However, writing synthesizable Register-Transfer Level code for such accelerators is a complex, expensive, and time-consuming process, which is cumbersome for early architecture development phases. To tackle this issue, a pre-synthesis simulation toolchain is herein proposed that facilitates the early architectural evaluation of complex accelerators aggregated to multi-level memory hierarchies. To demonstrate its usefulness, the proposed gem5-accel is used to model a tensor accelerator based on Gemmini, showing that it can successfully anticipate the results of complex hardware accelerators executing deep Neural Network models.**

***Index Terms*—Simulation Toolchain, Accelerator Modeling, Complete System Emulation**

I. Introduction

DURING the past decade, several contributions have been made to tackle the challenges involved in the conception of pre-Register-Transfer Level (RTL) development tools targeting heterogeneous high-performance processing systems, pointing out important problems, such as the lack of sufficiently accurate models and the scalability of existing solutions [1]. Although relevant proposals on design exploration and performance prediction were presented, they usually target fairly homogeneous systems (such as conventional multi-core Central Processing Unit (CPU) systems) [2], [3], and are hardly suited for heterogeneous systems featuring specialized hardware accelerators. Furthermore, they are also unsuited for early design exploration phases (where the parameterization of the architectures is not fixed yet), as they usually require a detailed specification of the device [4].

In contrast, this paper proposes gem5-accel, a toolchain based on the gem5 simulator [5] and gem5-ndp [6] targeting the fast modeling and evaluation of new heterogeneous architectures and their integration with processing systems and memory hierarchies without resourcing to complex and time-consuming RTL-description approaches. gem5-accel also offers a software library to easily devise the control and synchronization layer between the CPU and the hardware accelerators, as well as a Load-Store Unit (LSU) specially designed to cope with the existing CPU virtual memory system (making address translation transparent to the developer). The proposed tool provides an accurate simulation of the modeled accelerators (which can be tailored down to a clock-cycle-basis) in parallel with the remaining components of the system. In addition, it inherits the gem5 interface to McPAT [7] and Cacti [8], allowing to parameterize the modeled hardware and estimate area and energy metrics (not explored herein).

Work supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under project 2022.06780.PTDC, projects UIDB/50021/2020 (INESC-ID), UIDB/EEA/50008/2020, and EXPL/EEI-HAC/1511/2021 (Instituto de Telecomunicações), research grant SFRH/BD/144047/2019, and European High Performance Computing Joint Undertaking (JU) under Framework Partnership Agreement No 800928 and Specific Grant Agreement No 101036168 (EPI SGA2). The JU receives support from the European Union's Horizon 2020 research and innovation programme and from Croatia, France, Germany, Greece, Italy, Netherlands, Portugal, Spain, Sweden, and Switzerland.

To demonstrate the functionalities and benefits of gem5-accel, a Neural Network (NN) accelerator inspired on Gemmini [9] was modeled and evaluated by considering the execution of nine microbenchmarks and twelve full Convolutional Neural Network (CNN) models. The conducted experimental procedure showed that the developed gem5-accel-based model successfully reproduced the equivalent physical architecture of the targeted Gemmini accelerator, by reveling performance benefits similar to those measured in [9].

II. gem5-accel Toolchain

The proposed toolchain offers three main mechanisms to support the development and evaluation of new accelerators: (1) an **architectural model** that acts as a boil-plate for new custom architectures; (2) a **software library** to implement the control and synchronization layer between the CPU and the hardware accelerators; and (3) **simulation scripts** to easily connect accelerators with the remaining processing system, while still allowing to tune the system components.

A. The architectural model

The proposed architectural model comprises three main components (see Fig. 1): (1) a memory-mapped Programming Interface (PI) for control and synchronization between the CPU and the hardware accelerator; (2) an LSU to handle communication between the hardware accelerator and memory hierarchy; and (3) a virtual memory translation mechanism to automatically convert memory addresses between virtual and physical domains without needing extra hardware.

The PI serves as a communication interface between the CPU and an accelerator for control purposes. The number and size of registers in the PI can be configured in gem5-accel, and their purpose is determined by the developer based on the custom architecture's requirements. As illustrated in Fig. 2, when the CPU accesses PI registers, the PI module automatically handles the request, using the `readPI` and `writePI` methods to read or write the programming registers. Importantly, the PI does not place any restrictions on the internal architecture of the accelerators, remaining unaware of their control flow, implementation, and Instruction Set Architecture (ISA).

The LSU features a memory port that connects the hardware accelerator to the memory hierarchy, and implements specific mechanisms for easy retrieval and storage of data. When exchanging data with memory devices, which impose request size constraints, the LSU automatically breaks down large requests into smaller ones that comply with such constraints (using the `accessMemory` method). These smaller requests are then sent one after the other at a configurable rate (via the `sendData` method). Additionally, to handle situations where memory devices are busy and reject requests, the LSU uses a First-In-First-Out (FIFO) request queue to retry sending the dropped requests when the memory device becomes available again.

Another important feature of the devised LSU is its ability to be coupled to any level of the memory hierarchy (or even multiple levels), with the bandwidth allowed at each level being a configurable

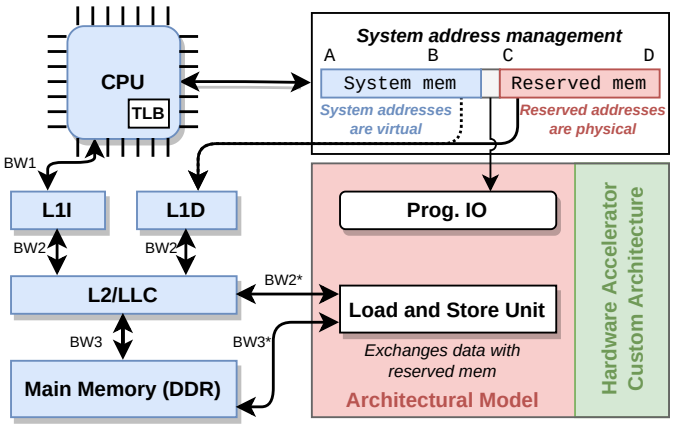

Fig. 1: Block-diagram of the proposed gem5-accel. BW2* and BW3* represent configurable parameters that define the bandwidths between the different memory hierarchy levels and the hardware accelerator.

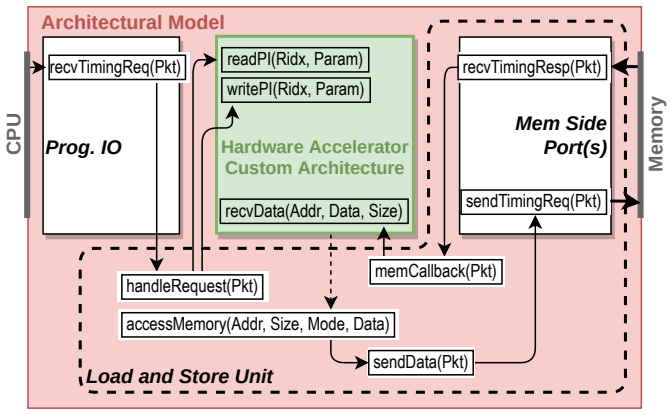

Fig. 2: Function diagram showing the relation between the software routines and the corresponding physical components depicted in Fig. 1.

parameter. gem5-accel also allows bypassing the memory controller, enabling developers to freely adjust the latency of the memory devices. This feature is particularly interesting for Near-Data Processing (NDP) design exploration [10], [11], allowing developers to emulate memory accesses using a custom latency. In addition, the architectural model of gem5-accel also supports multiple hardware accelerators coupled to the same or different CPUs and memory devices.

Due to the existence of a virtual memory subsystem, the data pointers made visible to user applications (virtual addresses) usually differ from those used to communicate with the memory devices (physical addresses). To make this compatible with the use of hardware accelerators, while avoiding the overheads of implementing explicit address translation mechanisms at the accelerator level, gem5-accel reserves a contiguous region within the existing physical memory and maps it into the virtual address space such that the virtual addresses are equal to the physical addresses within that region. Thus, that memory region can be shared between the CPU and the hardware accelerator without the need of explicitly translating the addresses. It is worth noting that this does not relax any security features, which are still enforced to guarantee process isolation, preventing other processes to access CPU-accelerator shared memory. In addition, this shared memory space has all the same features as the remaining memory, including being accessible by the CPU with the same latency. Hence, whenever required, it can also be used for other purposes.

B. Simulating hardware accelerators using gem5-accel

gem5-accel supports two simulation modes: **System Emulation (SE)** and **Full System (FS)**. In SE mode, it simulates a single application using the accelerator without an Operating System (OS). Hence, OS-specific features/restrictions are disabled. In contrast, FS mode simulates a standard OS (Linux), producing results similar to a real multitasking environment where the OS manages hardware resources and enforces process and memory isolation. As a result, applications can't directly access physical resources and the communication between the user application and the hardware accelerator has to be done through a device driver. Naturally, this introduces more complexity and programming overhead on the application side.

Listing 1 depicts a partial SE simulation script, highlighting the main components of the system and relevant configurations allowed by gem5-accel. The underlying processing system in this example features a standard CPU, a two-level cache hierarchy, and a Dynamic Random Access Memory (DRAM) (see Part 1). In Part 2, an accelerator is instantiated and its PI is mapped into the CPU address space. Part 3 connects the hardware accelerator to both the CPU and the memory hierarchy using configurable buses. By changing the width of these buses (in class `CustomL2XBar`) and the parameters of the memory components, it is possible to control the bandwidth made available to the accelerator, allowing to simulate different levels of proximity to the memory. Although this example shows the accelerator integrated with the L2 cache, it can be as easily coupled to the L1 or the DRAM.

Accordingly, a C code example using a hardware accelerator is shown in Listing 2. First, the data pointers representing both the accelerator PI registers and the shared memory region between the CPU and accelerator are set statically to the corresponding addresses. Then, both the operands and the kernel (list of accelerator-specific instructions) are loaded into memory and the CPU instructs the accelerator to start executing the kernel. While the accelerator is active, the CPU may execute other tasks. Finally, the CPU waits for the accelerator to finish processing by pooling its status register.

Listing 1: gem5 SE script simulating a standard processing system and memory hierarchy with a custom accelerator using gem5-accel. In this example, the accelerator is connected to the L2 cache. Nevertheless, gem5-accel allows to connect accelerators to any memory device.

```
# Standard processing system
system.cpu = X86MinorCPU()
system.cpu.icache = L1Cache()                                    (Part 1)
system.cpu.dcache = L1Cache()
system.l2cache = L2Cache()
system.mem_ctrl = DDR3_1600_8x8()

# Custom accelerator
system.accel = CustomAccel(
    ctrl=("0x40000000", "0x40001000"),                           (Part 2)
    data=("0x40001000", "0x80000000"),
    max_rsze=0x40, max_reqs=64)

class CustomL2XBar(L2XBar):
    def __init__(self):
        super(CustomL2XBar, self).__init__()
        width = 32  # 256-bit bus width (BW2*)

# Connect custom accelerator to CPU
system.accel.cpu_port = system.cpu.dcache_port                   (Part 3)
system.cpu.dcache.cpu_side = system.accel.mem_side

# Connect custom accelerator to L2
system.l2bus = CustomL2XBar()
system.accel.dma_port = system.l2b.slave
system.l2cache.cpu_side = system.l2bus.master

# Let accelerator operate on upper 1GB of a 2GB RAM
system.cpu.workload[0].map(
    0x40000000,  # Host address space
    0x40000000,  # Accelerator address space                     (Part 4)
    0x40000000,  # Address range
    cacheable=True)

exit_event = m5.simulate() # Start simulation
```

Listing 2: C code example to initialize and control a hardware accelerator using gem5-accel SE mode. The routines `initData`/`initKernel` preload the data/kernel into the shared memory region. The accelerator is instructed to start executing the kernel by `__accelLaunchKernel`, and `__accelReady` checks (pooling) if the accelerator finished processing.

```
int main() {
    // Assign accelerator memory addresses
    volatile void *accel_control = ACCEL_CRTL_ADDR;
    volatile DATA_TYPE *dataset = ACCEL_DATA_ADDR;
    volatile void *kernel = ACCEL_KRNL_ADDR;

    // Dataset and kernel are prepared
    initData(dataset); initKernel(kernel);

    // CPU launches kernel on accelerator
    __accelLaunchKernel(accel_control);

    ...  // CPU processes tasks in background

    // CPU waits for accelerator to finish
    while (!__accelReady(accel_control));

    ...} // CPU post-processes the results
```

III. Case Study: A Gemmini-based CNN Accelerator

To validate the proposed toolchain, a Gemmini-based NN accelerator inspired in [9] was modeled and evaluated using gem5-accel. This accelerator (see Fig. 3) consists of a bi-dimensional systolic array, where each Processing Element (PE) contains a Multiply-Accumulate (MAC) unit and communicates with its neighboring PEs through dedicated buses. Additionally, PEs can be arranged in tiles, which communicate with adjacent tiles through pipeline registers. It also includes specialized units to perform matrix transposition, rearrangement of input matrices for convolution, Rectified Linear Unit (ReLU), accumulation, bit-shift, pooling, and matrix-scalar multiplication, as well as dedicated input/output memories (scratchpad and accumulator SRAM) explicitly managed by the CPU.

A. Experimental Setup

In the conducted experiments, the modeled accelerator consists of a systolic mesh of 16-by-16 PEs with a 256 kB scratchpad memory and a 64 kB accumulator output memory. The remaining processing system consisted of: an in-order superscalar CPU (similarly to [9]) operating at 2 GHz; a two-level cache hierarchy with a 32 kB L1I/D cache and a 256 kB L2 cache; and a 2 GB DDR3 memory.

Three sets of benchmarks were executed using the aforementioned setup: (1) nine microbenchmarks consisting of common NN operations; (2) twelve CNN models of the darknet framework [12] (inference only); and (3) three hand-tuned CNN models optimized for the modeled Gemmini accelerator. Additionally, the microbenchmarks of experiment (1) were also executed using an official Gemmini emulation platform supported on Verilator and the results were compared with those of gem5-accel. For simplicity, the experiments were conducted using SE mode. Nevertheless, gem5-accel also supports to FS mode.

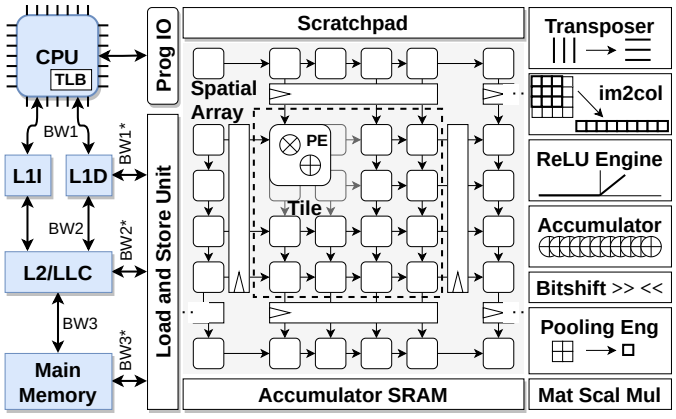

Fig. 3: Generic architecture of a Gemmini NN accelerator consisting of a configurable systolic mesh of PEs grouped into tiles, as well as NN-specific units and input and output memories.

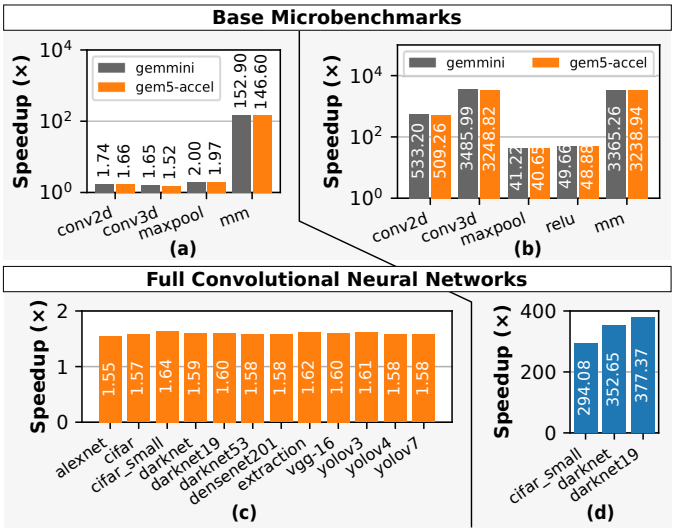

Fig. 4: Microbenchmarks (a, b) and CNN benchmarks (c, d). (a, c) regard applications where the gathering of the operands is done by the CPU. (b, d) concern applications fully executed by the Gemmini accelerator.

B. Experimental Results and Discussion

The results obtained with the first experiment are illustrated in Fig. 4a and Fig. 4b. The nine considered microbenchmarks are divided in two groups: (a) kernels whose operands are gathered by the CPU and arranged in the scratchpad in the order they are required for the operations; and (b) kernels that are fully executed by the accelerator.

In the first scenario, the CPU replaces the functionality of the `im2col` and `transposer` blocks of the Gemmini accelerator. Naturally, this leads to a larger memory footprint in the scratchpad (as the convolution operands are stored in a redundant fashion), and to a significant increase in execution time (since more data is transferred from memory and the performed accesses are irregular). Hence, the benchmarks of the first group attain a much lower speedup than their counterparts of the second group, which are exclusively executed by the Gemmini accelerator, including the gathering of the operands. It is also worth noting that when comparing the performance results of gem5-accel (orange bins) with those of Gemmini's official simulation platform (gray bins), an average error as low as 4.3 % can be observed, supporting the accuracy of gem5-accel and the developed model.

The second experiment considers twelve complete CNN models from the darknet framework. However, due to the way darknet is organized (which benefits execution in CPUs and GPUs), offloading the gathering of the operands to the accelerator would require complex modifications. Thus, only the convolution and pooling operations were offloaded to the accelerator, while the gathering of the operands was done by the CPU. Naturally, this limited the achievable performance improvements, which ranged from 1.55× to 1.64×, as shown in Fig. 4c.

Nevertheless, this experiment shows the usefulness and robustness of gem5-accel, allowing to execute a benchmark as complex as darknet in a realistic processing system featuring a custom hardware accelerator and a communication layer between the host code and the accelerator.

Fig. 4d shows the speedup of three hand-tuned CNNs optimized to fully exploit the capabilities of the modeled accelerator. By offloading

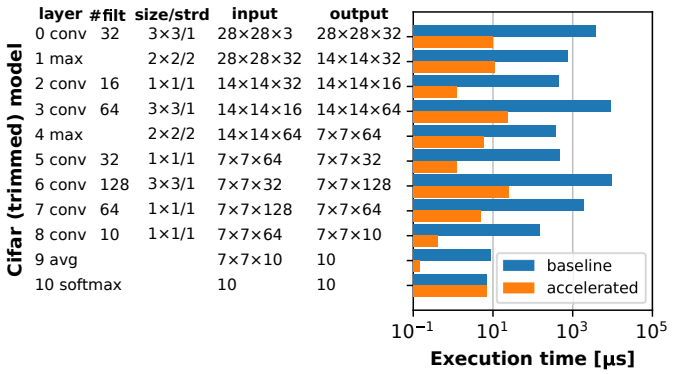

Fig. 5: Execution time of each layer of a small CNN optimized for the CIFAR-10 dataset when executed only by the CPU (blue bars) and the CPU equipped with the hardware accelerator (orange bars).

the gathering of the operands to the accelerator, speedups of more than two orders-of-magnitude are achieved, as reported in [9]. In particular, Fig. 5 depicts the decrease of the execution time of convolutional and pooling layers for a small CNN targeting the CIFAR-10 dataset.

IV. Related Work

Recently, two works have been proposed targeting design space exploration of heterogeneous hardware accelerators that have gained particular visibility in the community: gem5-Aladdin [13] and gem5-SALAM [14]. Although they share similar goals with gem5-accel, these approaches are crucially different in the adopted simulation strategies (see Table I), which is reflected in their scope and scalability.

Both solutions employ a High-Level Synthesis (HLS) approach, starting with developers describing a behavioral model of the accelerators using high-level C code and structures. Then, the application code is analyzed, and a Dynamic Data Dependence Graph (DDDG) is built, with vertices being Low Level Virtual Machine (LLVM) Intermediate Representation (IR) instructions and edges representing operation dependencies. Finally, operations are automatically scheduled to the accelerators to maximize resource utilization and performance.

Although the high-level hardware modeling enabled by these tools makes it easier for the user to quickly develop and evaluate new accelerators, it also limits the complexity of their internal architectures, which is supported by the rather simple benchmarks used to validate these works. In contrast, gem5-accel is comparatively lower-level, allowing the user to describe new accelerators using a syntax closer to the hardware, enabling to model much more complex (and even micro-programmed) co-processors, such as Gemmini [9].

Furthermore, gem5-Aladdin and gem5-SALAM involve multi-step simulation procedures, which can make their adoption more difficult. gem5-Aladdin even requires analytical steps, which may introduce significant errors to the results, compared with a pure-simulation approach like gem5-accel. Additionally, neither of these tools produces gem5 simulation objects, with the architecture of the simulated accelerators being unknown to gem5. Thus, generating gem5 statistics for the internal operations of these components is not possible.

Another advantage of gem5-accel over these two tools is its support for both SE and FS modes. While gem5-Aladdin only supports SE mode (with a custom virtual memory system), gem5-SALAM only supports bare-metal FS mode and has no virtual memory support. On the other hand, gem5-accel provides full support to virtual memory using standard mechanisms, making it transparent to the user.

Finally, all three solutions are compatible (to some extent) with energy and area estimation mechanisms. gem5-Aladdin provides a custom model for energy estimation. gem5-SALAM supports energy and area analysis through native integration with McPAT/Cacti. gem5-accel, however, requires extra hardware information to be able to create area and energy profiles of the modeled accelerator using McPAT/Cacti together with traces generated by gem5.

TABLE I: Summary and comparison of main features offered by gem5-accel and two previous works: gem5-Aladdin [13] and gem5-SALAM [14].

	gem5-Aladdin	gem5-SALAM	gem5-accel
Development tier (analogy)	HLS	HLS	RTL description
Simulation flow	Multi-step, analytical analysis (requires the use of multiple tools and analytical models	Multi-step	Single-step
Performance simulation	Trace-based	Clock-cycle/ interval-based	Clock-cycle/ interval-based
Energy estimation	Custom model	McPAT/Cacti support	Possible using McPAT/ Cacti gem5 integration
Area estimation	Not supported	McPAT/Cacti support	Possible using McPAT/ Cacti gem5 integration
Requires rebuilding gem5	No (architecture of accelerators unknown to gem5)	No (architecture of accelerators unknown to gem5)	Partial rebuild (accelerators are gem5 components)
Address translation	Custom	Not supported	Standard (w/ particular address mapping)
Simulation mode	SE only	FS bare-metal/ limited support to FS with SO	SE/FS

V. Conclusions

In this paper, gem5-accel, a pre-RTL simulation toolchain allowing to easily model, test, and estimate the performance attained by new hardware accelerators without resourcing to complex and time-consuming RTL-description workflows is proposed.

To demonstrate the devised toolchain, three experiments were performed, that led to two important conclusions: (1) gem5-accel allows to easily predict the benefits of entire processing systems with custom accelerators executing full applications such as darknet [12]; and (2) gem5-accel provides a very accurate simulation of custom hardware accelerators, as shown by the similarity of the results obtained with gem5-accel and Gemmini's official simulation platform.

References

[1] M. Herget, et al. Design space exploration for distributed cyber-physical systems: State-of-the-art, challenges, and directions. In *DSD*. IEEE, 2022.
[2] G. Mariani, et al. OSCAR: an optimization methodology exploiting spatial correlation in multicore design spaces. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2012.
[3] R. Jongerius, et al. Analytic multi-core processor model for fast design-space exploration. *IEEE Trans. Computers*, 2018.
[4] G. Singh, et al. NAPEL: near-memory computing application performance prediction via ensemble learning. In *DAC*. ACM, 2019.
[5] N. L. Binkert, et al. The gem5 simulator. *SIGARCH CAN*, 2011.
[6] J. Vieira, et al. gem5-ndp: Near-data processing architecture simulation from low level caches to DRAM. In *SBAC-PAD*. IEEE, 2022.
[7] S. Li, et al. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, 2009.
[8] S. J. E. Wilton and N. P. Jouppi. CACTI: an enhanced cache access and cycle time model. *IEEE J. Solid State Circuits*, 1996.
[9] H. Genc, et al. Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration. In *DAC*. IEEE, 2021.
[10] J. D. Ferreira, et al. pLUTo: Enabling massively parallel computation in DRAM via lookup tables. In *MICRO*. IEEE, 2022.
[11] E. Lockerman, et al. Livia: Data-centric computing throughout the memory hierarchy. In *ASPLOS*. ACM, 2020.
[12] J. Redmon. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/, 2013–2016.
[13] Y. S. Shao, et al. Co-designing accelerators and soc interfaces using gem5-aladdin. In *MICRO*. IEEE Computer Society, 2016.
[14] S. Rogers, et al. gem5-salam: A system architecture for llvm-based accelerator modeling. In *MICRO*. IEEE, 2020.