IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 18 August 2022, accepted 29 August 2022, date of publication 5 September 2022, date of current version 15 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3204664

RESEARCH ARTICLE

Architecture and Application Co-Design for Beyond-FPGA Reconfigurable Acceleration Devices

ANDREW BOUTROS[1,2], (Student Member, IEEE), ERIKO NURVITADHI[2], (Member, IEEE), AND VAUGHN BETZ[1], (Fellow, IEEE)

[1]Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada
[2]Programmable Solutions Group, Intel Corporation, Santa Clara, CA 95054, USA

Corresponding author: Andrew Boutros (andrew.boutros@mail.utoronto.ca)

This work was supported in part by the Intel/VMware Crossroads 3D-FPGA Academic Research Center and in part by the NSERC/Intel Industrial Research Chair in Programmable Silicon.

ABSTRACT In recent years, field-programmable gate arrays (FPGAs) have been increasingly deployed in datacenters as programmable accelerators that can offer software-like flexibility and custom-hardware-like efficiency for key datacenter workloads. To improve the efficiency of FPGAs for these new datacenter use cases and data-intensive applications, a new class of reconfigurable acceleration devices (RADs) is emerging. In these devices, the FPGA fine-grained reconfigurable fabric is a component of a bigger monolithic or multi-die system-in-package that can incorporate general-purpose software-programmable cores, domain-specialized accelerator blocks, and high-performance networks-on-chip (NoCs) for efficient communication between these system components. The integration of all these components in a RAD results in a huge design space and requires re-thinking the implementation of applications that need to be migrated from conventional FPGAs to these novel devices. In this work, we introduce RAD-Sim, an architecture simulator that allows rapid design space exploration for RADs and facilitates the study of complex interactions between their various components. We also present a case study that highlights the utility of RAD-Sim in re-designing applications for these novel RADs by mapping a state-of-the-art deep learning (DL) inference FPGA overlay to different RAD instances. Our case study illustrates how RAD-Sim can capture a wide variety of reconfigurable architectures, from conventional FPGAs to devices augmented with hard NoCs, specialized matrix-vector blocks, and 3D-stacked multi-die devices. In addition, we show that our tool can help architects evaluate the effect of specific RAD architecture parameters on end-to-end workload performance. Through RAD-Sim, we also show that novel RADs can potentially achieve 2.6× better performance on average compared to conventional FPGAs in the key DL application domain.

INDEX TERMS Deep learning, field-programmable gate arrays, hardware acceleration, network-on-chip, reconfigurable computing.

I. INTRODUCTION

Field-programmable gate arrays (FPGAs) have been continuously growing in capacity and heterogeneity over the past decades. Besides their *soft* programmable logic and routing, FPGA fabrics now include a wide variety of embedded *hard* blocks such as on-chip memories, fracturable multi-precision multipliers and high-speed transceivers to enhance their efficiency [1]. However, with the increasing deployment of FPGAs as datacenter accelerators, we are witnessing a more radical transition from conventional FPGAs to more complex *beyond-FPGA* reconfigurable acceleration devices (RADs). These are heterogeneous devices that integrate a traditional reconfigurable fabric with other forms of

The associate editor coordinating the review of this manuscript and approving it for publication was Vincenzo Conti.

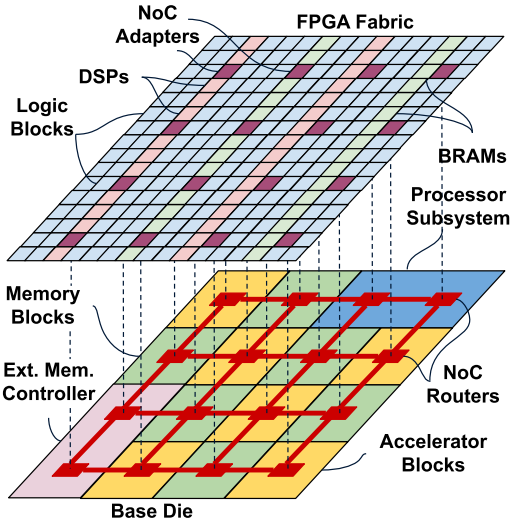


FIGURE 1. Example 3D-stacked RAD instance integrating an FPGA fabric and an ASIC base die with different accelerator blocks, more on-chip memory, external memory controllers, and a general-purpose processor subsystem.

compute architectures that can range from general-purpose Turing-complete scalar processors to coarse-grained domain-specialized massively-parallel accelerator blocks. The backbone of these RADs is one or multiple high-performance packet-switched networks-on-chip (NoCs) which provide efficient system-level communication between the different RAD components and a gateway to high-bandwidth device IOs (e.g. to external memory, Ethernet, PCIe). An example of such RADs is the Xilinx Versal architecture that combines general-purpose ARM cores, vector processors for deep learning (DL) acceleration and an FPGA fabric, all connected via a pervasive system-wide NoC [2].

The recent advances in passive and active interposer technology also enable the realization of multi-die system-in-package RADs, in which the system-level NoC(s) act as a continuous communication layer across dice or even between distinct devices. Fig. 1 depicts an example RAD instance using 3D chip integration technology, such as Intel Foveros [3], to stack an FPGA fabric on top of an ASIC base die. In this example, the base die contains different NoC-connected accelerator blocks, larger on-chip memories and other hardened components such as external memory controllers and processor subsystems. Application modules that require frequent changes and customization are implemented on the reconfigurable FPGA fabric and can access the NoC to communicate with other RAD components through hard fabric NoC adapters that connect to the base die routers via interposer micro-bumps.

Design of such complex devices is challenging; an architect not only needs to select many design parameters for each independent component of a RAD, but must also consider the complex interactions between these different components. This results in a multiplicatively larger design space to explore for architecting novel and efficient RADs. What makes the design problem even more complicated is that FPGA applications can not be effortlessly migrated to novel RADs or readily make the best use of the specialized accelerator blocks and NoC-based communication for improved performance. Therefore, architects need to re-think the implementation of applications while designing their novel RADs, which creates a more challenging architecture and application co-design problem.

The design of conventional FPGA fabrics has been extensively studied with well-established research tools for exploring and evaluating new architecture ideas, such as the Verilog-to-Routing (VTR) flow [4]. These tools help answer questions on how to best architect the fine-grained programmable routing fabric and logic blocks, what type of hard blocks to integrate in the fabric, and the effect of these architecture enhancements on FPGA computer-aided design (CAD) algorithms and compile time. However, these tools are inadequate for RAD architecture exploration as they lack the following desired qualities:

1) **Application-driven:** These tools focus on optimizing FPGA architectures based on application-agnostic performance metrics such as the maximum operating frequency of given benchmark circuits. For complex RADs with coarse-grained accelerator blocks and latency-insensitive NoCs, architecture exploration must be driven by end-to-end application-specific performance. In other words, the key metric is how fast a given application is executed on a candidate RAD (cycles or runtime) rather than how fast a given circuit is clocked on an FPGA fabric (clock frequency).
2) **Higher level of abstraction:** Conventional FPGA architecture exploration is typically driven by applications written in a hardware description language (HDL), which can create a productivity bottleneck when re-designing applications for RADs.
3) **Rapid design space exploration:** FPGA application designers usually rely on register-transfer level (RTL) simulation for functional verification of their applications. For RAD architecture and application co-design, RTL simulation would be very slow for such large complex systems and would require developing a tremendous amount of system components in HDL such as NoC routers, accelerator blocks, memory controllers, etc. This labour-intensive approach would significantly limit the turn-around time for RAD architecture exploration, especially at early stages of the design process.
4) **Packet Routing:** Mapping application designs to a conventional FPGA architecture involves placing logic blocks and routing wires between them on the programmable fabric. It has no notion of packet-switched NoC-based communication between modules which is the backbone of novel RADs.

Our work addresses this gap by introducing RAD-Sim, a cycle-level architecture simulator for rapid application-driven architecture exploration of RADs. It allows architects to perform what-if analysis to study the complex interactions

between various design choices in a RAD, and facilitates the architecture and application co-design process via its higher-level design abstraction. RAD-Sim takes as inputs SystemC descriptions of application modules and other RAD components (e.g. specialized accelerator blocks or scalar processor cores), detailed NoC specifications, and module NoC placement constraints. It performs detailed system-level simulation and produces end-to-end application performance results and NoC traffic reports for the application's traffic patterns. It can also be used to verify the application's functionality when given appropriate test vectors and their expected golden outputs. This enables architects to write RAD applications at a high-level of abstraction and rapidly tweak them as needed for architecture exploration while verifying their functionality during this process, and also to evaluate the effect of specific RAD architecture parameter choices or application re-design decisions on the overall end-to-end performance.

In our prior work [5], we introduced RAD-Sim as a tool for rapid architecture exploration of novel RADs incorporating FPGA fabrics, coarse-grained accelerator blocks and high-performance system-level NoCs. To further showcase the utility of RAD-Sim in the co-design of architecture and applications for RADs, this paper's contributions are:

- Presenting a case study of migrating a state-of-the-art DL application, the neural processing unit (NPU) overlay, from conventional FPGAs to different RADs. This study shows how RAD-Sim highlights performance bottlenecks and provides insights to guide the architecture and application co-design process for RADs.
- Introducing a new bandwidth-aware design approach for optimizing the NPU when mapped to RADs with embedded NoCs.
- Studying a variety of RADs ranging from multi-die NoC-connected FPGAs to devices augmented with specialized matrix-vector accelerator blocks, and multi-active-die 3D-stacked architectures.
- Showcasing novel 3D-stacked RADs that can achieve 2.6× higher performance on average when compared to current conventional FPGAs with up to 145 TOPS effective performance on key DL workloads.

We also open-source RAD-Sim along with the NPU example design to facilitate future research on novel RAD architectures.[1]

II. BACKGROUND AND RELATED WORK

A. THE EMERGENCE OF BEYOND-FPGA RADs

The recent large-scale deployments of FPGAs in datacenters, pioneered by the Microsoft Catapult project [6], [7], have highlighted two key use cases that FPGAs excel at. Firstly, FPGAs are used as *bump-in-the-wire* devices to pre/post-process data streams on-the-fly by performing tasks such as network security, packet processing and data compression, freeing up CPU resources for the core compute of datacenter workloads. For example, the Pigasus project [8] uses a single FPGA to perform 100 Gbps network intrusion detection and prevention, reducing the CPU core count requirement to only 5 cores instead of 364 cores in the software-only solution. Other work shows that FPGA-based smart network interface cards (NICs) can increase the efficiency of distributed DL training system by up to 2.5× when accelerating the inter-node all reduce communication and gradient compression, freeing up CPU resouces for the core compute-intensive tensor operations [9]. Also, as the number of solid-state drives (SSDs) per server increases, FPGAs can also perform near-data processing in SmartSSDs to alleviate the processor-to-storage bandwidth bottleneck [10].

Secondly, the network-connected datacenter FPGAs can be flexibly combined into datacenter-scale *service accelerators* that offer low-latency processing for key datacenter services at a fraction of the power budget as in Microsoft's Brainwave for DL inference [11] and Bing's search engine [6]. In both use cases, processing pipelines are frequently changed or upgraded, which justifies the use of FPGAs as they offer faster time-to-solution and less development effort compared to taping out specialized fixed-function chips. In addition, FPGAs also offer a variety of high-bandwidth IOs that enable efficient data steering at the crossroads between different datacenter server endpoints such as network, storage, CPU cores and accelerators.

However, the FPGA's fine-grained programmable routing fabric is struggling to keep up with the ever-increasing FPGA transceiver bandwidth and data flow of key datacenter workloads [12]. To mitigate these challenges, prior academic research has shown that embedding hard packet-switched NoCs in FPGA fabrics can offer tremendous on-chip data steering bandwidth at a minimal area cost and without affecting the FPGA's flexibility [13], [14]. As a result, hard NoCs were recently adopted in commercial FPGAs from Xilinx [15], Achronix [16], and Intel [17]. Besides their programmable routing and logic, modern FPGAs incorporate a variety of hardened ASIC-style blocks that ideally capture common functionalities across as many applications as possible without sacrificing the FPGA's flexibility. Taking DL acceleration as an example, the composition of layers, data manipulation between them, vector operations, and pre/post-processing stages might significantly differ between different workloads, which can benefit from the FPGA's reconfigurability. However, all of them include many dot-product operations that can benefit from the increased efficiency of hardening as high-performance tensor cores [18], [19], [20].

These trends in FPGA architecture along with recent advances in 2.5D and 3D chip integration technologies [3], [21] have resulted in the emergence of a new class of beyond-FPGA reconfigurable devices that combine the flexibility of FPGAs, the efficiency of hard NoCs for data steering, and the high-performance of specialized accelerator blocks. Our work focuses on building tools that enable rapid architecture exploration for these complex devices and also

[1]Code can be downloaded at: https://github.com/andrewboutros/rad-flow

facilitate application re-design when migrating from conventional FPGAs to novel RADs.

B. FPGA ARCHITECTURE EXPLORATION

One unique property of FPGAs is their generality; they can be reconfigured to implement a variety of applications, some of which might not even exist at the FPGA's design time. Therefore, FPGA architects have to evaluate their new architectural ideas using a carefully curated set of benchmark circuits that capture key FPGA application domains. Major FPGA vendors have their own proprietary sets of internal benchmarks and customer designs for optimizing their products. However, prior academic work has compiled suites of representative benchmarks to be used for architecture exploration of both general-purpose [22] or domain-optimized [23] FPGAs. These benchmark circuits are written in an HDL such as Verilog or VHDL, and are mapped (i.e. synthesized, placed and routed) to a given FPGA fabric architecture using a retargetable CAD system.

VTR [4] is an academic open-source FPGA CAD flow that is widely used for FPGA architecture exploration and CAD research. It takes as inputs an architecture description defining the FPGA blocks, routing architecture, and their area/timing models, along with a set of benchmarks to be mapped to this architecture. This flow can be used to evaluate different FPGA architecture candidates based primarily on application-agnostic quality metrics such as the maximum operating frequency of benchmark designs or the silicon footprint of low-level FPGA circuitry.

However, this conventional FPGA architecture exploration flow is not sufficient for evaluating RAD architectures that include other complex components besides the traditional FPGA fabric (e.g. NoCs and hard accelerator blocks). For example, it cannot evaluate the end-to-end performance of applications with some components implemented in the FPGA fabric and others executed on instruction-controlled accelerator blocks, or produce key system-level metrics such as NoC congestion and traffic patterns. On the other hand, although standalone NoC simulators do exist [24], they lack features to simulate a coupled FPGA fabric and cannot fully evaluate a RAD architecture.

C. ARCHITECTURE SIMULATORS

Architecture simulators are widely used to guide architectural decisions, especially during early stages of the design process. Depending on their level of detail, they can provide fast and accurate performance estimates without the need for detailed RTL implementation of architecture ideas. To drive architecture research, many simulators with different features and areas of focus have been introduced to facilitate design space exploration of classic von Neumann architectures as well as specialized accelerators and emerging compute technologies. `gem5` is arguably the most commonly used CPU simulator in architecture research [25]. It performs high-fidelity cycle-level modeling of modern CPUs and can run full applications for different instruction set architectures. More recently, `gem5` added support for modeling GPUs based on the AMD Graphics Core Next architecture [26]. Over the years, it has been used to evaluate many computer architecture research ideas such as [27], [28], [29]. Other CPU architecture simulators have been introduced such as Sniper [30] and XIOSim [31] with different foci on more scalable multi-core CPU simulation and more accurate performance and power modeling of mobile cores, respectively. GPGPU-Sim [32] is another academic simulator for contemporary Nvidia GPU architectures that can run CUDA or OpenCL workloads and supports advanced features such as TensorCores and CUDA dynamic parallelism. Unlike these examples, our work does not target classic von Neumann architecture exploration, but rather focuses on novel RADs that combine traditional FPGA fabrics with other styles of compute architectures. To evaluate RAD architectures, the input to the simulator is not just compiled application instructions. Instead, it can be a mix of instructions for any software-programmable RAD components (e.g. coarse-grained accelerator blocks) and custom user-defined modules implemented on the FPGA fabric.

Many simulators are also implemented to evaluate custom application-specific accelerator architectures such as in [33], [34], and [35]. Aladdin [36] is a more general accelerator simulator for estimating the performance and power of specialized dataflow hardware from a high-level C description. More recently, gem5-Aladdin [37] integrates the `gem5` CPU simulator with Aladdin to model systems-on-chip (SoCs) that include both CPUs and accelerator functional units with the main focus on system-level considerations such as memory interfaces and cache coherency. Similarly, our RAD-Sim can model specialized accelerator blocks as components of a RAD architecture. However, it accepts any user-specified accelerator design written in SystemC and is not limited to dataflow accelerators controlled by finite-state machines as in Aladdin. RAD-Sim also combines accelerator blocks with other application modules implemented on the RAD's reconfigurable fabric and with packet-switched NoCs for system-level communication; evaluating such combined systems is not possible in gem5-Aladdin.

SIAM [38] is a recent example of an architecture simulator focusing on emerging compute technologies. It models chiplet-based in-memory compute for deep neural networks, and integrates architecture, NoC, network-on-package, and DRAM models to simulate an end-to-end system. Although our work similarly aims to model complete systems integrating different components including NoCs and specialized accelerator blocks, it is not limited to only modeling in-memory DL compute and focuses mainly on the reconfigurable computing domain. For modeling RADs, another key difference is that both the placement of compute modules and their attachment to NoC routers have to be flexible (i.e. not an architecture choice but programmed at application design time) due to the reconfigurability of the FPGA fabric.

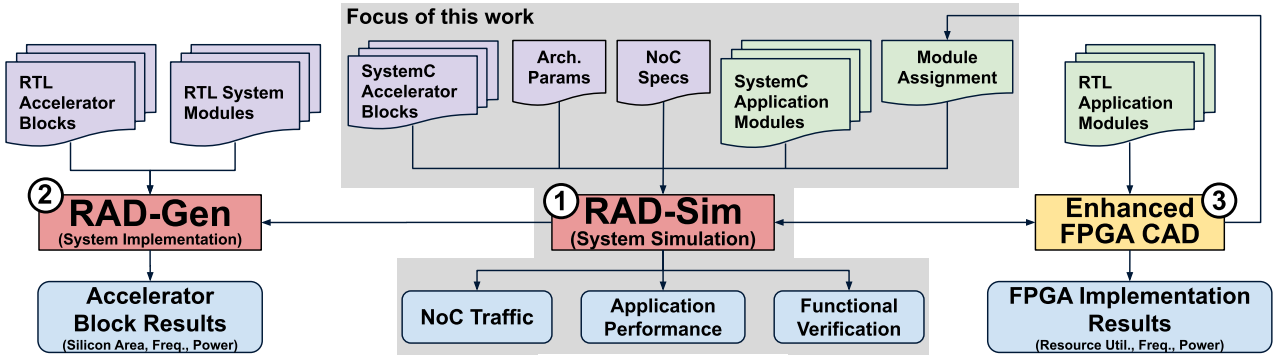


FIGURE 2. Overview of our RAD architecture exploration and evaluation flow. This paper focuses on the RAD-Sim component (highlighted in grey).

III. RAD ARCHITECTURE EXPLORATION FLOW

In this section, we will first introduce an overview of our complete architecture exploration flow for RADs. Then, we will focus only on the first component of our flow, RAD-Sim, which is used to perform initial rapid architecture exploration and evaluation. The other components of the flow will be covered in future works.

A. FLOW OVERVIEW

Fig. 2 gives an overview of our complete RAD architecture exploration and evaluation flow. The first component of this flow is **RAD-Sim**, an architecture simulator for rapidly exploring the design space of RADs and studying the complex interactions between their different components. The inputs of RAD-Sim are a set of RAD architecture parameters, specifications for one or more system-level NoCs, and SystemC descriptions of the RAD accelerator blocks. These accelerator blocks can be any latency-insensitive circuit with an AXI-compatible interface that the designer would like to experiment with hardening in a RAD architecture. This allows designers to experiment with accelerator blocks that have different functionalities, granularity and programming models (e.g. instruction-controlled, finite state machine, fixed pipeline) and also to compose different styles of accelerator blocks in a RAD. In addition, the user provides application-related inputs which are SystemC descriptions of the application design modules implemented on the FPGA and placement constraints assigning their ports to specific NoC adapters throughout the FPGA fabric. Then, RAD-Sim performs system simulation to estimate end-to-end performance and produce NoC traffic reports. To verify the functionality of the application design mapped to a given RAD instance, a user can pass input test vectors and their expected outputs, which can be extremely useful when RADs and applications are co-designed during the early stages of architecture exploration.

After narrowing down the design space to a few candidate architectures, more detailed evaluation can be conducted using the other two components of our flow. Given the RTL implementations of the specialized accelerator blocks as well as other system components such as the NoC routers and any other hardened functionalities, **RAD-Gen** pushes these modules through the ASIC implementation flow to provide architects with silicon area footprint, timing, and power results for these blocks. Both RAD-Sim and RAD-Gen will share the same front-end that takes as an input the RAD architecture parameters and NoC specifications. RAD-Gen will then modify a parameterizable NoC router implementation based on the user-specified inputs. It will then push the RTL implementations of the NoC and other system modules through existing ASIC implementation tools targeting either proprietary standard cell libraries or open-source ones (e.g. FreePDK [39] and OpenRAM [40]). To perform power analysis of a RAD/application combination, RAD-Gen will be used to obtain energy per operation results for the implementation of the RAD's ASIC components on a given process technology. These results, coupled with toggle rates/activities collected by RAD-Sim for a specific simulated application, can be used to estimate the overall power consumption. On the other hand, an **enhanced FPGA CAD** flow is used to synthesize, place, and route the application design modules to be implemented on the reconfigurable fabric of a candidate RAD. An enhanced version of the VTR flow (in development) can directly model NoC routers/adapters as hard blocks embedded in the FPGA fabric. However, we can also model them in commercial CAD tools by creating reserved logic locked regions of appropriate size and locations, and connecting design module interfaces to registers placed in these regions.

Additionally, the embedding of hard NoCs in FPGA fabrics presents a new placement problem as modules must be placed not only where they have sufficient fabric resources and minimize traditional programmable routing delay, but also so that their connection to NoC adapters on nearby routers does not cause undue NoC congestion. RAD-Sim can evaluate NoC performance (latency and congestion) given a specific placement solution and expected application NoC traffic patterns. The enhanced FPGA CAD tools can then use these metrics to adjust module placement and assignment to NoC adapters/routers, and iterate again if latency constraints are not met. This is similar in concept to invoking static timing analysis during the placement stage in the conventional

FPGA CAD flow to evaluate the expected critical paths of a design in order to guide optimization. While in this work we focus mainly on hard NoCs, RAD-Sim can also readily model application designs that include soft NoCs either as a design component or a pre-placed and routed interconnect overlay such as [41], [42].

This paper focuses mainly on the first component of this flow, RAD-Sim, and its use for both rapid architecture exploration and architecture-application co-design for RADs.

B. RAD-SIM IMPLEMENTATION DETAILS

RAD-Sim is developed in SystemC, which allows designers to model their hard accelerator blocks and application modules at various levels of abstraction, trading off model faithfulness for designer productivity. For example, a specific module can be described using SystemC in a high-level behavioral way for fast development time, or a more detailed (closer to RTL) way that can be input to high-level synthesis tools to generate hardware. RAD-Sim uses BookSim [24] to perform cycle-accurate NoC simulation. BookSim is an open-source NoC simulator that has been leveraged by many other system simulators, such as GPGPU-Sim. It is heavily parameterized to allow modeling a wide variety of interconnect networks with different topologies, routing functions, arbitration mechanisms, and router micro-architectures. It is also easily extendable to support other features that are not provided out-of-the-box depending on specific use cases.

RAD-Sim builds on top of BookSim in three aspects:

1) It adds a SystemC wrapper around BookSim to allow combining the NoC with different accelerator blocks and application modules modeled in SystemC.
2) It complements BookSim by tracking packet contents to enable functional verification of actual applications on RADs. This is necessary because BookSim primarily focuses on performance estimation and hence models the arrival times of packets, not their contents. As we show later in Section IV, migrating an application design from a conventional FPGA to a new RAD instance can require significant re-architecting of the application to use the RAD NoC(s) for inter-module communication. Therefore, it is necessary to ensure that functionality is preserved during this process.
3) It implements SystemC NoC adapters that allow RAD architects to experiment with different user-facing NoC abstractions, independently of the underlying NoC protocol and physical implementation details.

The NoC adapters implemented in RAD-Sim also perform clock domain crossing and width adaptation between the application modules or hard accelerator blocks and the NoC. For example, we provide users with AXI streaming (AXI-S) and AXI memory-mapped (AXI-MM) adapters, but RAD-Sim is structured to be modular such that architects can implement their custom or standardized NoC adapter protocol of choice and easily integrate it in the simulator. Fig. 3 shows the AXI-S master and slave NoC adapters implemented in RAD-Sim as an example. They consist of

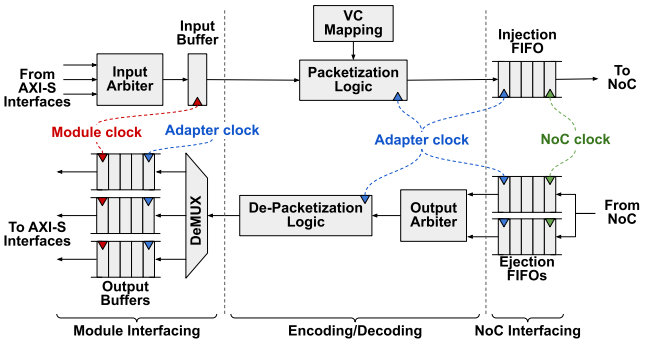


FIGURE 3. AXI streaming slave (top) & master (bottom) NoC adapters implemented in RAD-Sim.

three main stages: module interfacing, encoding/decoding, and NoC interfacing. For the slave adapter, an input arbiter selects one of the (possibly multiple) AXI-S interfaces connected to the same NoC router. Once an AXI-S transaction is buffered, it is packetized into a number of NoC flits and mapped to a specific NoC virtual channel (VC). Then, these flits are pushed into an asynchronous FIFO to be injected into the NoC depending on the router channel arbitration and switch allocation mechanisms. The master adapter works in a similar way but in reverse: flits are ejected from the NoC and once a tail flit is received, they are depacketized into an AXI-S transaction which is then steered to its intended module interface. The adapters implemented in RAD-Sim are parameterized to allow experimentation with different arbitration mechanisms, VC mapping tables, and FIFO/buffer sizes. They also support up to three distinct clock domains where the connected module, adapter, and NoC can be all operating at different clock frequencies. This enables experimentation with scenarios where stages of the NoC adapters are either hardened or implemented in the FPGA's soft logic.

Table 1 lists some of the user input parameters of RAD-Sim. Besides these parameters, RAD-Sim takes as an input a NoC placement file that specifies the assignment of all hard accelerator block and fabric module ports to specific NoC routers/adapters. This is currently passed as a user-specified manual assignment; in our future work we also plan to enable automatic creation of this file such that NoC latency constraints specified by the user are met and/or overall application performance is optimized. As described in Sec. III-A, this router assignment file could be automatically created by an enhanced FPGA placement algorithm that repeatedly adjusts the routers to which modules connect (essentially placing the router interfaces) as placement proceeds and invokes RAD-Sim to quantify the effect of these adjustments on the system performance.

In addition, RAD-Sim provides users with various telemetry utilities to record specific simulation events and traces along with different scripts to visualize the collected data. This can be very useful in reasoning about the complex interactions between the different components of a RAD and understanding the effect of changing various architecture parameters on the overall application performance. Fig. 4

TABLE 1. RAD-Sim user input parameters.

User Input	Description
`rad_instance`	Specify whether monolithic or 3D device
`num_nocs`	No. of system-wide NoCs
`noc_payload_width`	Bit width of NoC links for flit payload
`noc_freq`	NoC operating frequencies
`noc_topology`	NoC typologies (e.g. mesh, torus)
`noc_dim`	NoC dimensions (for certain topologies)
`noc_routing_func`	NoC routing algorithm (e.g. dim. order)
`noc_vcs`	No. of NoC virtual channels
`noc_vc_buffer_size`	Depth of virtual channel buffers (words)
`noc_out_buffer_size`	Depth of router output buffers (words)
`noc_packet_types`	No. of different packet types
`noc_router_uarch`	Router micro-arch (e.g. input queue)
`noc_vc_allocator`	Router VC allocation mechanism
`noc_sw_allocator`	Router switch allocation mechanism
`noc_credit_delay`	Delay for sending back NoC credit
`noc_routing_delay`	Delay for calculating packet route
`noc_vc_alloc_delay`	Delay for router VC allocation
`noc_sw_alloc_delay`	Delay for router switch allocation
`adapter_fifo_size`	Depth of adapter ejection/injection FIFOs
`adapter_obuff_size`	Depth of adapter output buffer (words)
`adapter_in_arbiter`	Adapter input arbitration mechanism
`adapter_out_arbiter`	Adapter output arbitration mechanism
`adapter_vc_mapping`	Mapping of flit types to virtual channels
`adapter_freq`	Adapter operating frequencies
`module_freq`	Accelerator or app. module frequencies
`sim_driver_freq`	Frequency of the simulation driver
`log_verbosity`	Level of simulation logging verbosity
`num_traces`	No. of event traces recorded
`trace_names`	Identifiers of recorded event traces

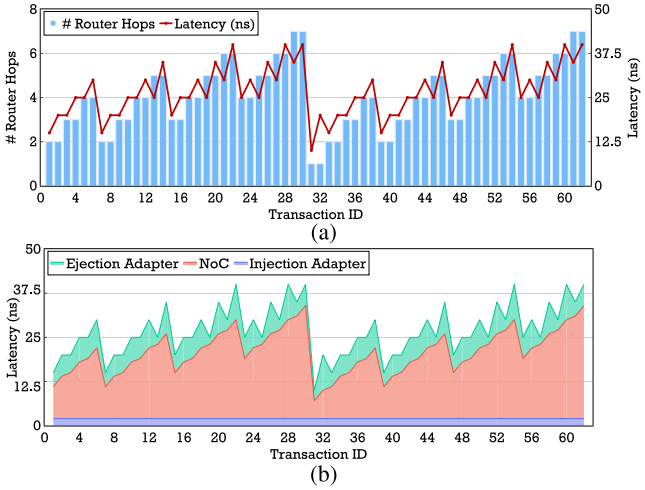


FIGURE 4. Example visualizations from RAD-Sim for an unloaded 4 × 4 mesh NoC showing: (a) Overall communication latency, number of hops, and (b) Latency breakdown.

shows example visualizations produced by RAD-Sim when trying to characterize the unloaded communication latency for a RAD with a 4 × 4 mesh NoC and two modules connected to each router. In this example experiment, a single module sends two AXI-MM transactions to the first module connected to each router (15 routers × 2 transactions) one at a time, with no other traffic on the NoC. This then repeats for the second module connected to each router. The module, adapter and NoC operating frequencies are set to 200 MHz, 800 MHz, and 1 GHz, respectively. The RAD-Sim telemetry utilities are used to record various timestamps in the transaction lifetime such as transaction initiation at the source module, packetization, injection/ejection, depacketization, and receipt at the destination module. Fig. 4(a) shows the latency in nanoseconds and number of NoC router hops for each of the 62 issued transactions. The graph shows how the number of hops and communication latency increase as the distance between the source and destination modules increases then drops when moving to the next row in the 4 × 4 mesh of routers. Fig. 4(b) shows another visualization produced by RAD-Sim that breaks down the latency for each transaction into time spent in the injection adapter, the NoC, and the ejection adapter. This can highlight the overhead introduced when experimenting with different adapter implementations.

IV. CASE STUDY: RE-DESIGNING THE NPU FOR RADs

In this section, we present a case study to showcase the capabilities of RAD-Sim by migrating a state-of-the-art DL FPGA benchmark, the NPU, from conventional FPGAs to novel RADs. This study highlights how RAD-Sim can pin-point performance bottlenecks and allows rapid experimentation with potential solutions both by re-designing the application to better suit RAD architectures and by changing the parameters of the RAD architecture itself.

A. THE NEURAL PROCESSING UNIT (NPU) OVERLAY

In this section, we present a brief overview of the NPU overlay that we use as a vehicle for our case study. The NPU is a state-of-the-art FPGA *soft processor* (i.e. software-programmable processor implemented on an FPGA's programmable fabric) with an instruction set and compute pipeline specialized for the acceleration of memory-intensive DL models such as multi-layer perceptrons (MLPs), recurrent neural networks (RNNs), gated recurrent units (GRUs), and long short-term memory models (LSTMs). The NPU architecture is similar to that of the Microsoft Brainwave architecture [11] and achieves an order of magnitude higher performance on Intel's DL-targeted FPGA, the Stratix 10 NX, when compared to same-generation GPUs [43].

Fig. 5 depicts the NPU overlay architecture which consists of several coarse-grained compute blocks chained together such that the output of one block is forwarded to the next. The key block in the NPU architecture is a massively parallel matrix-vector multiplication unit (MVU). It consists of T tiles, each of which has D sets of C dot-product engines (DPEs) of length L multiplication lanes. Each DPE is tightly coupled with a register file (RF) that stores all the model weights persistently on-chip and makes use of the tremendous on-chip bandwidth of the FPGA's BRAMs. An MVU tile computes a row block of a matrix-vector multiplication operation, and then their partial results are reduced and accumulated over multiple time steps (if needed) to output the final MVU result. This is followed by an external vector register file (eVRF) to skip the MVU for instructions that do not include a matrix-vector multiplication, and then two identical vector elementwise multi-function units (MFUs) for

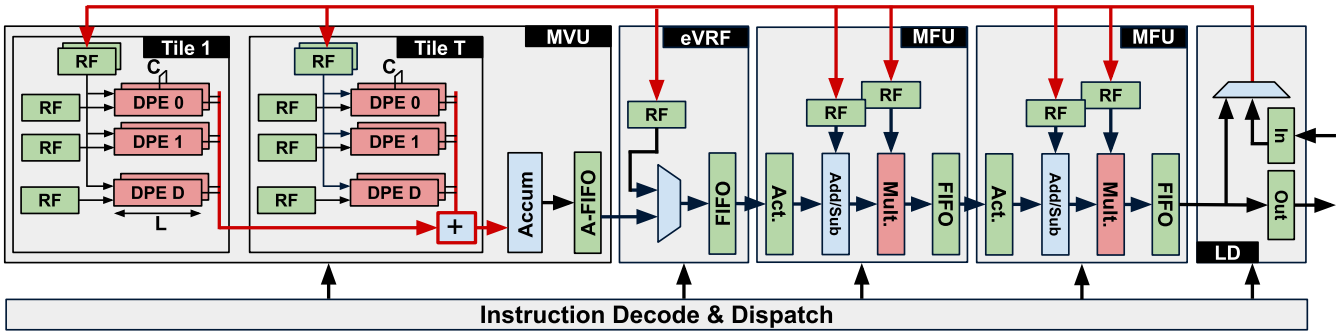


FIGURE 5. Overview of the NPU overlay architecture consisting of five chained coarse-grain compute blocks: a matrix-vector multiplication unit (MVU), an external vector register file (eVRF), two identical vector elementwise multi-function units (MFUs), and a loader (LD) block. The connections highlighted in red are latency sensitive channels.

operations such as activation functions, addition/subtraction, and multiplication. Finally, there is a loader block (LD) which writes back the pipeline results to any of the NPU's architecture states and communicates with other external modules or interfaces. The NPU processor datapath is massively parallel to target the highly parallel and regular DL computations in contrast to general-purpose processor pipelines. For instance, an NPU with 2 cores, 7 tiles, 40 DPEs, and 40 lanes can execute up to 45,000 operations in a single cycle.

All these blocks are controlled by very long instruction words (VLIW). Each field of the VLIW is decoded into a sequence of micro-operations and dispatched to its corresponding compute block by a central control unit as shown in Fig. 5. The NPU overlay is heavily optimized and compiled once to generate a single bitstream that is deployed on an FPGA and then programmed purely through software to run different applications. An NPU compiler has also been developed to compile a Keras TensorFlow model description into NPU VLIW instructions that execute on the FPGA. We refer interested readers to [44] and [43] for more details about the NPU architecture and front-end.

B. BASELINE SystemC NPU MODEL

We implement SystemC simulation models for the different NPU blocks to use them in RAD-Sim as either hard accelerator blocks or fabric application modules. To evaluate the faithfulness of our SystemC NPU model, we compare it to cycle-accurate RTL simulation of the NPU SystemVerilog implementation. We use Synopsys VCS v2016.06 for the RTL simulations, and run both the SystemC and RTL simulations on the same 24-core Intel Xeon Gold 6146 CPU. We use an NPU configuration similar to that in [43] with 2 cores, 7 tiles, 40 DPEs and 40 lanes, which we also use for the rest of our experiments in this paper.

We run simulations for a variety of NPU workloads including simple matrix-vector multiplications (GEMV), RNNs, GRUs, LSTMs, and MLPs of different sizes, and report the results in Fig. 6 in tera operations per second (TOPS). The results show that our SystemC simulation model can estimate NPU performance to a high degree of accuracy with average

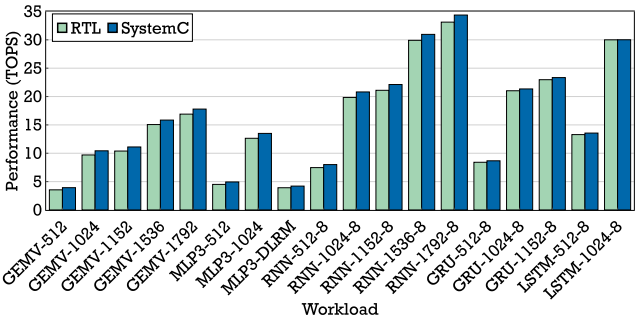


FIGURE 6. NPU performance results from RTL and SystemC simulations. Our SystemC NPU model estimates performance with less than 10.8% error (5.1% on average) and is 26× faster than RTL simulation.

error of only 5.1% and maximum error of 10.8% compared to cycle-accurate RTL simulation. These small differences result from minor discrepancies between our SystemC and RTL implementations of the NPU architecture that can be tuned to further reduce this gap. However, the SystemC simulations are 26× faster than the RTL simulations on average, with speedups ranging from 6.5× to 100× depending on the workload size. The speed of SystemC models contributes to the larger architecture space we can explore in RAD-Sim, and the close agreement in performance results between the SystemC and RTL simulation means we can trust the RAD-Sim results to have high fidelity for this case study.

C. FULLY LATENCY-INSENSITIVE AXI-STREAMING NPU

To map the NPU to a RAD instance incorporating a NoC, all communication channels between NPU blocks have to be latency-insensitive (LI). All the feedforward communication between the five chained NPU blocks already contains elastic FIFO interfaces. However, there are two main latency sensitive channels that need to be modified (highlighted in red in Fig. 5). The first is the connection from the LD block to all the different RFs which is used for writing back the pipeline results and issuing instruction tag updates for data hazard resolution. The second is the inter-tile reduction connections between all T tiles and the accumulator within the MVU. Since the MVU alone constitutes 52%, 77% and

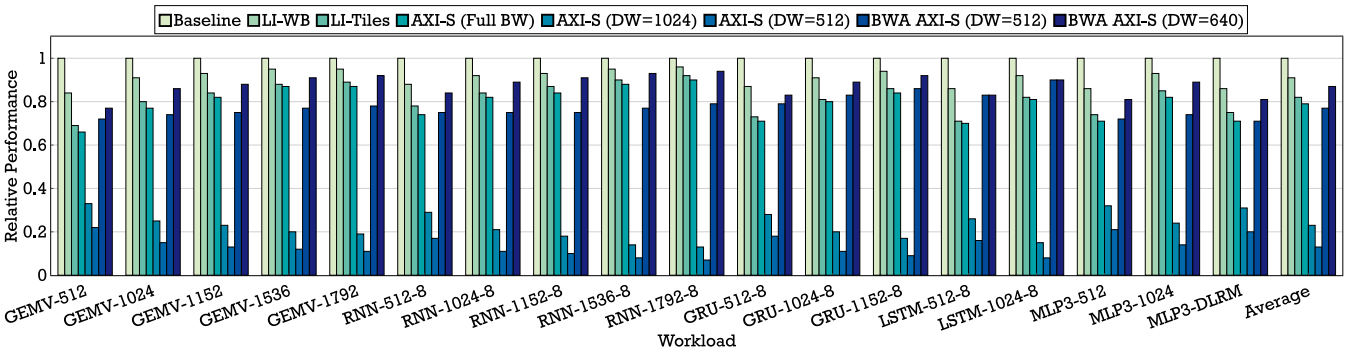


FIGURE 7. Relative performance of NPU design iterations for migration from a latency sensitive FPGA design to a latency-insensitive, NoC-ready design. All results are on a conventional FPGA. The figure compares: (1) baseline NPU, (2) adding LI write-back, (3) adding finer granularity LI for MVU tiles and accumulator, (4) adding AXI-S wrappers without bandwidth limitations, (5) limiting the AXI-S interfaces data width to 1024 and 512 bits, and (6) using a bandwidth-driven design approach (BWA) with 512-bit and 640-bit AXI-S interfaces. The results show that direct migration can significantly degrade performance (AXI-S with DW = 512) and therefore requires careful re-architecting of the application design (BWA AXI-S).

78% of the NPU's logic, BRAM and tensor block (TB) resources respectively, it is desirable to have finer granularity latency-insensitivity such that the different tiles and the accumulator can be treated as distinct modules when mapped to RADs.

We first add an elastic FIFO for the write-back output of the LD block and change the latency-sensitive instruction tag update broadcast signal to sequential point-to-point LI messages from the LD block to each of the other blocks individually. Then, we also break the inter-tile latency sensitive reduction chain by adding an elastic output FIFO as well as a separate instruction FIFO and issue logic for each of the T tiles and the accumulator. After this change, all tiles send their outputs to the accumulator block which then performs both the reduction and accumulation operations. Finally, we implement AXI-S wrappers around each of the NPU modules so we can connect them to the AXI-S NoC adapters in RAD-Sim. These wrappers implement the AXI-S protocol on top of the FIFO interfaces and are parameterized to allow for changing the data width of the AXI-S interfaces. These parameterized AXI-S wrappers decouple the compute of these modules from the communication bandwidth between them, allowing us to study the effect of bandwidth restrictions on the overall NPU performance.

Fig. 7 shows the simulation results for the effect of these changes on the NPU performance when running our benchmarks. On average, the performance is degraded by 9% when changing the write-back and hazard resolution channel to be LI. This is mainly due to the sequential tag update messages from the LD block to each of the other NPU blocks. Then, an additional 9% performance hit results when we add finer granularity LI to the MVU tiles and accumulator, for two reasons. Firstly, the number of sequential tag update messages that have to be sent by the LD block for data hazard resolution increases from four (MVU, eVRF, 2 MFUs) to ten (7 tiles, eVRF, 2 MFUs). Secondly, the latency for the accumulator to reduce the tile outputs increases and it also does not start execution until the outputs of all 7 tiles are ready to be consumed. Since most of the NPU workloads have strict sequential dependencies (e.g. between layers in MLPs or between time steps in RNNs, GRUs, LSTMs), increasing latency results in pipeline bubbles, reducing performance. Finally, adding the AXI-S wrappers causes less than 3% performance degradation when they are set to the full widths of the NPU block interfaces.

Now that the NPU is fully LI, we can map it to RAD architectures where communication and computation are decoupled by a NoC. As a start, we map the LI AXI-S-wrapped NPU modules to a simple RAD with only an FPGA fabric and an ideal (unrealistic) NoC. This ideal NoC implements point-to-point connections between the NPU modules without any additional arbitration or latency due to traversing multiple NoC links and has no bandwidth contention between different traffic streams traversing the NoC at the same time. We experiment with NoC routers with 1024-bit and 512-bit interfaces. Although this limits the inter-module communication bandwidth between the NPU modules compared to the baseline design, these router interface widths are not unrealistic; 512-bit interfaces are a common design choice for NoC adapters in prior academic research [13] and in the Xilinx Versal NoC architecture [45]. Even in the case of an idealized NoC, however, this significantly throttles the NPU performance to only 23% and 13% of the original performance for interface widths of 1024 and 512 bits, respectively. This experiment highlights that migrating application designs as-is from FPGAs to novel RADs with embedded NoCs can lead to very poor performance; instead migration requires careful consideration of inter-module communication bandwidth.

D. BANDWIDTH-DRIVEN DESIGN APPROACH

Fig. 8(a) shows a graph representation of the LI NPU architecture where nodes represent different NPU modules and edges are communication channels between them. Each edge is annotated with the channel bit width as a relation to NPU architecture parameters (C and D are number of cores and DPEs introduced in Sec. IV-A) and the numbers in brackets represent the bit width for the NPU configuration we use with $C = 2$ and $D = 40$. It is clear that the NPU was originally designed to exploit the tremendous amount of on-chip programmable interconnect bandwidth with very

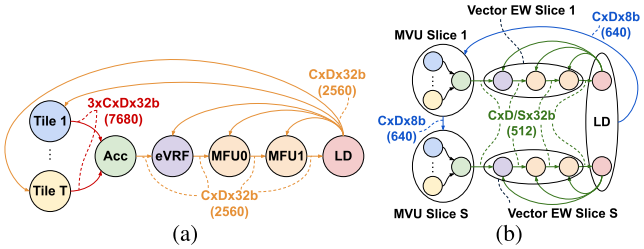


FIGURE 8. Graph representation of inter-module communication in (a) LI NPU and (b) re-structured LI NPU using bandwidth-driven design.

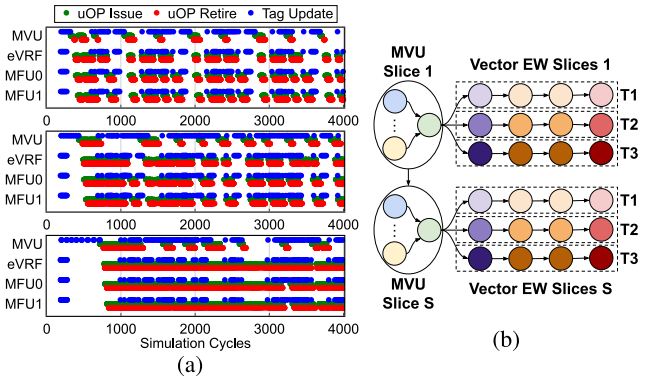


FIGURE 9. (a) RAD-Sim traces for an NPU slice with 1 thread (top), 2 threads (middle), and 4 threads (bottom), and (b) Graph representation of the multi-threaded NPU. The expensive MVU slices are multi-threaded and the cheaper vector elementwise slices are replicated to consume the MVU outputs of different thread executions in parallel.

wide busses of 7680 and 2560 bits between different modules. However, these extremely wide interfaces are not friendly for communication over a NoC with limited router interface widths. Therefore, the boundaries between NPU modules need to be re-structured in a way that limits the widths of edges in Fig. 8(a). We refer to this as a *bandwidth-driven design* approach; it follows three principles:

1) Where possible, convert high-bandwidth inter-module channels into local intra-module ones to use the more abundant wires in an ASIC implementation (of accelerator blocks) or on the FPGA fabric, rather than crossing module boundaries over the NoC.
2) Split modules that consist of independent parallel compute lanes into finer-granularity modules to limit the input/output bandwidth of each module.
3) Divide broadcast channels communicating with different modules that use different data widths into separate channels to avoid unnecessary data padding and transfer.

The graph representation in Fig. 8(b) shows how we re-structure the NPU architecture following these bandwidth-driven design principles. Firstly, since each tile consists of D independent DPEs that send their results to the accumulator block to be reduced with the corresponding D DPEs from other tiles, we split each tile into S groups of $\frac{D}{S}$ DPEs each. Then, we combine the corresponding groups from different tiles with a smaller accumulator in a new module that we refer to as an *MVU slice*. With 5 MVU slices ($S = 5$), the output interface of each slice is limited to only 512 bits at the cost of replicating the instruction FIFO and a few of the tile RFs for each slice. This adds negligible logic and increases NPU memory utilization by less than 9%. Secondly, since all other NPU blocks (eVRF, MFUs, LD) are operating as independent single-instruction multiple-data (SIMD) lanes, we also split them into vector elementwise (EW) slices of $\frac{D}{S}$ SIMD lanes to match the MVU slice output bandwidth.

Thirdly, we split the unified broadcast LD write back channel into multiple channels with data widths matching that of the destination modules. The LD block internally combines the results from different vector EW slices only when writing back to the MVU slices; otherwise each LD slice writes back to its corresponding vector EW slice modules independently as shown in Fig. 8(b). The data width of the MVU write back channel is set to match the `int8` numerical precision of the MVU since it is now an independent channel and does not talk to the other blocks (eVRF and MFUs) using `int32` precision. This limits the width of this channel to 640 bits at no additional cost. Finally, to parallelize MVU write backs or tag updates for the NPU's data hazard resolution, we also add message-passing channels from one MVU slice to the next. By doing this, the LD can send only one write back or tag update message to the first MVU slice; this message is then passed between slices and the LD can start sending messages to other NPU blocks in parallel.

The results in Fig. 7 show that, with the same amount of compute resources and an AXI-S interface data width restricted to 512 bits, the bandwidth-driven re-structuring of the NPU can gain back most of the performance lost to bandwidth limitations. It even exceeds the performance of the original NPU (that used very wide, latency-sensitive communication) due to the added parallelism in tag updates through the MVU slice-to-slice message passing channels. The total cost of the NPU re-design to be fully LI and have bandwidth-friendly 512-bit AXI-S interfaces is an average 23% degradation in performance compared to the original latency-sensitive NPU in [43]. With a slight increase in AXI-S interface width to 640 bits, performance increases by 10% on average due to matching the full width of the LD write-back and MVU slice-to-slice communication channels. This brings the fully LI NPU to within 87% of the original NPU performance on average, as shown in Fig. 8(b).

E. NPU MULTI-THREADING

After restructuring the NPU to be more modular, LI, and bandwidth-friendly as described above, we experiment with mapping it to a realistic NoC. We again assume a RAD instance with a conventional FPGA fabric (similar to a Stratix 10 NX) and no accelerator block, but this time we use a realistic 9×9 mesh NoC. For this experiment, we use 512-bit AXI interfaces for all the NPU modules. We assume the restructured NPU modules run at 300 MHz similar to

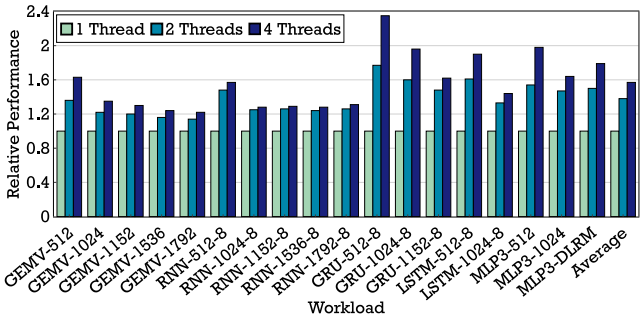


FIGURE 10. Relative performance of multi-threaded NPU with 1, 2 and 4 threads mapped to an FPGA with an embedded NoC in RAD-Sim.

the original NPU, with the NoC adapters and routers running at 1.2 GHz and 1.5 GHz, respectively. The NoC has 166-bit wide links (for flit payload and meta data), 3 VCs, input queuing router architecture, and uses dimension order packet routing. We also use the same settings for our experiments with different RAD examples detailed in Section V.

We perform a manual module assignment of an NPU with five slices ($S = 5$) to NoC routers and pass it as an input to RAD-Sim. Each MVU slice receives its instructions/inputs and sends its outputs through a separate router, while each vector EW slice receives its instructions and inputs from its corresponding MVU slice through another dedicated router. In addition, both the combined LD interface and the central instruction dispatch unit are connected to their dedicated routers to communicate with the rest of the NPU. This module assignment utilizes 12 out of the 81 NoC routers.

We simulate the NPU workloads in RAD-Sim to evaluate the effect of the additional inter-module communication latency through the NoC. On average, the overall performance decreased by 9% compared to the LI bandwidth-friendly NPU using the ideal NoC from Sec. IV-D, which translates to 0.68× the performance of the original latency-sensitive NPU on Stratix 10 NX. The top simulation trace in Fig. 9(a) produced using RAD-Sim's telemetry utilities shows the first 4000 cycles of a single NPU slice running the GRU-512 workload as an example. The green, red and blue circles represent micro-operation (uOP) issue, retire and tag update events in each of the NPU blocks. There are long idle gaps in the MVU slice simulation trace due to sequential dependencies on both previous GRU time step results and vector operations in the current time step; NoC latency has exacerbated these gaps.

This data highlights an opportunity: interleaving the execution of multiple instruction streams (i.e. *threads*) in the MVU slice could fill these idle gaps. Fig. 9(b) illustrates the graph representation of an NPU architecture with support for three interleaved thread executions. The MVU slices switch from one thread execution to another while waiting for sequential dependencies to be resolved, and direct the output of each thread to a different set of vector EW slices (labeled $T1$, $T2$, $T3$ in the figure). The middle and bottom traces in Fig. 9(a) show the reduction in idle gaps when the NPU supports two and four interleaved thread executions, respectively. Fig. 10 shows that interleaving two and four threads can increase the overall performance by 38% and 57% on average (and up to 77% and 135%) respectively, vs. a single thread implementation. However, adding support for each additional thread utilizes 17%, 23% and 20% more ALMs, BRAMs and TBs, respectively. Therefore, it is not feasible to implement more than one thread on the (already full) Stratix 10 NX 2100 used by the baseline NPU. Nevertheless, it is feasible to implement more threads when exploring RADs with bigger/multiple FPGA fabrics or hard accelerator blocks that free up more fabric resources, as we discuss in the next section.

TABLE 2. Specifications of example RADs used in our study.

	FPGA Sectors	ASIC Sectors	Integration Tech.	NPU Config.
RAD1	8×5 (×2)	–	2.5D passive	1 × 4-thread
RAD2	8×5	2×5	Monolithic	1 × 4-thread
RAD3	8×5	8×5	3D active	4 × 1-thread

V. NPU ON RADs

In the previous section, we have shown that RAD-Sim can highlight performance bottlenecks and help architects experiment with application re-design ideas (e.g. bandwidth-driven restructuring and multi-threading for our NPU example) to alleviate these bottlenecks. In this section, we will illustrate how RAD-Sim can capture a variety of RAD architectures by mapping the NPU to three example RAD instances ranging from a multi-die FPGA using passive interposers to a monolithic FPGA with side accelerator complex and a device using 3D active die stacking. Additionally, we will show how RAD-Sim can be used to fine-tune specific architecture parameters and quantify the effect on end-to-end performance. The intention of the experiments presented in this section is by no means to perform a detailed architecture study to find the best RAD architecture for a specific application, which is an ongoing work combining the use of both the RAD-Sim and RAD-Gen components of our flow. Instead, we aim to illustrate that RAD-Sim can capture a wide variety of RAD styles and also guide the fine-tuning of low level architecture parameters of these devices.

Use of LI bandwidth-driven design (as illustrated for the NPU in the previous section) and a system-level NoC completely decouples the application compute from its inter-module communication. This raises the interconnect abstraction level and enables the exploration of complex RADs that span multiple dice and incorporate hard accelerator blocks. In this case, the conventional FPGA CAD tools do not need to optimize the timing and routability of signals crossing the boundaries between dice through interposers or trying to reach the programmable routing interfaces of a hard accelerator block. If each application module meets timing separately and can be connected to a NoC adapter, the evaluation of end-to-end application performance on a given RAD instance is raised to the cycle-level simulation of soft/hard modules and NoC latency; this is exactly what is captured by RAD-Sim.

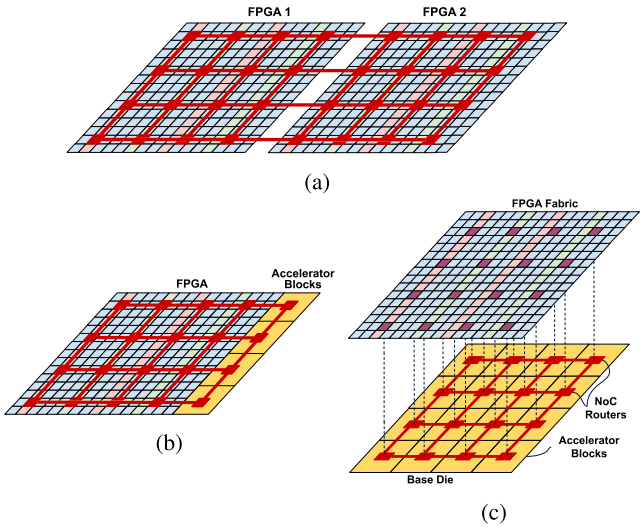


FIGURE 11. RADs used in our study: (a) 2.5D integration of 2 FPGA fabrics with the NoC links crossing through a passive interposer, (b) monolithic FPGA fabric and coarse-grained hard accelerators, and (c) 3D integration of an FPGA fabric on top on an ASIC die of accelerator blocks.

A. EXPERIMENTAL SETUP & METHODOLOGY

We map the re-designed LI AXI-S multi-threaded NPU to three example RADs and evaluate their performance using RAD-Sim. The first device (RAD1), illustrated in Fig. 11(a), consists of two identical FPGA fabrics using 2.5D chip integration [46] where the NoC links are the only wires crossing from one fabric to another through a passive interposer. The second one (RAD2) in Fig. 11(b) is a monolithic device with an FPGA fabric and a separate complex of hard accelerator blocks. In this case, the accelerator blocks can only be accessed from the fabric using the NoC. Finally, the third device (RAD3) is an FPGA fabric 3D-stacked on top of a base die of accelerator blocks as shown in Fig. 11(c). 3D-stacked RADs offer distributed access to more NoC routers, thereby avoiding the congestion of specific links when several modules on the reconfigurable fabric are trying to access a side accelerator complex, for example. We define the term *FPGA sector* as a region of FPGA resources with a NoC router/adapter at its center. For example, an FPGA with 8×5 sectors has a total of 40 NoC routers/adapters throughout its fabric regardless of the NoC topology used (i.e. it does not have to be a regular 8×5 mesh topology). Equivalently, we define an *ASIC sector* as an area of silicon that has the same footprint of an FPGA sector and includes a hard accelerator block (possibly with other hardened components) and a NoC router. Table 2 summarizes the specifications of the three example RADs we use in our study.

For all three example RADs, we begin with an FPGA fabric with the same resources as an Intel Stratix 10 NX 2100 device (702k ALMs, 6,847 M20K BRAMs, 3,960 TBs). For RAD3, we assume that the FPGA fabric and the base die are implemented in the same process technology for simplicity. We remove resources to make room for any NoC routers added. We implement matrix-vector multiplication units that resemble the MVU slices of the NPU described in Sec. IV-D as the hard accelerator blocks in RAD2 and RAD3. These blocks are realistic candidates for hardening since they implement common functionality across almost all DL workloads, while the rest of the NPU blocks could be specialized for different workloads to increase efficiency [47] and thus benefit from the FPGA's reconfigurability. For RAD1, we use one FPGA for implementing the NPU's MVU slices (FPGA1) and the other FPGA to instantiate vector EW slices to support 4 interleaved thread executions (FPGA2). For RAD2 and RAD3, we first stamp out enough single thread NPU copies to utilize all the available hard accelerator blocks, and then use any remaining FPGA resources to add support for as many interleaved thread executions as possible. Table 2 lists the NPU configurations used.

TABLE 3. Resource utilization for the NPU portions implemented on the RAD FPGAs. (TBs: tensor blocks, M20K: 20Kb BRAMs.)

	ALMs	M20Ks	TBs
RAD1 - FPGA1	83,585 (12%)	5,260 (77%)	2,800 (71%)
RAD1 - FPGA2	494,216 (70%)	5,644 (82%)	3,200 (81%)
RAD2	550,0930 (78%)	2,632 (90%)	3,200 (81%)
RAD3	550,0930 (78%)	2,632 (90%)	3,200 (81%)

We estimate performance by using RAD-Sim to map the NPU to the three example RADs. We set an FPGA fabric operating frequency of 300 MHz (matching the NPU operating frequency in [43]) and conservatively assume that the hard accelerator blocks run at 600 MHz. We scale the operating frequency of the 28nm NoC routers from [48] to 1.5 GHz in 14nm process technology, and we assume that the NoC adapters operate at $4\times$ the fabric speed, similarly to [48]. The RTL implementation of the NoC router used in [48] is heavily parameterizable and compatible with Booksim parameters (developed by the same developers of Booksim). More details about this RTL implementation and its source code can be found in [49]. In all experiments, we use a mesh NoC topology (dimensions specified in Table 2 for each case) with 166-bit links, 3 VCs, input queuing router architecture, and dimension order packet routing. The depths of the NoC adapters' injection/ejection FIFOs and ouptut buffers (see Fig. 3) are set to 16 and 2, respectively. We also manually assign NPU module AXI-S ports to specific routers in a reasonable (but possibly sub-optimal) placement.

B. FPGA AND ASIC AREA RESULTS

To determine FPGA resource utilization, we synthesize, place and route the parts of the NPU to be implemented on the FPGA fabrics using Intel Quartus Prime Pro 21.2 on a Stratix 10 NX 2100 device. We use reserved logic lock regions at the appropriate locations for NoC routers/adapters, mark them as empty design partitions, and connect the NPU modules to them based on our manual module assignment to different routers. We conservatively size each logic lock region as a grid of 10×10 logic array blocks (LABs)

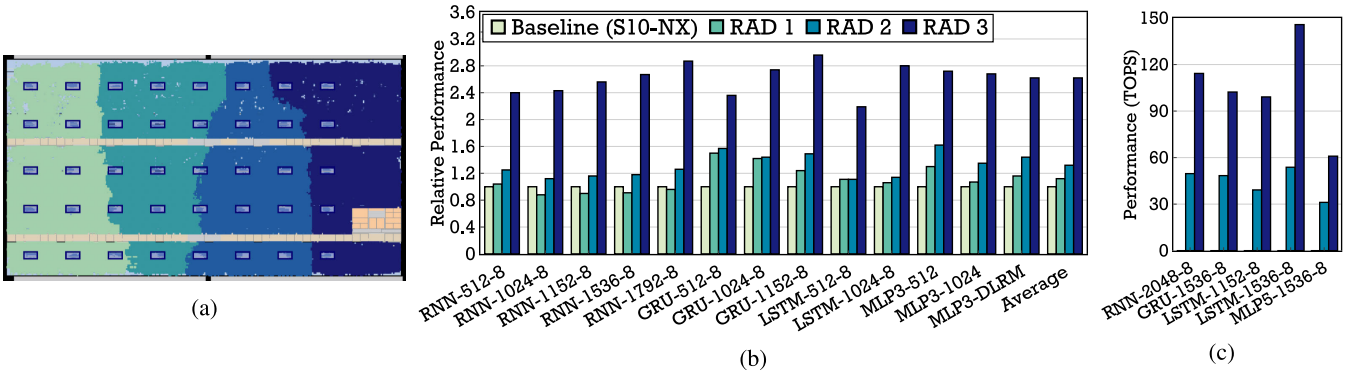


FIGURE 12. **(a) Quartus chip planner view of the FPGA fabric of RAD3. Different colors show the 4 instantiations of NPU vector elementwise modules working with hard MVUs on the base die. The small boxes are reserved logic lock regions for the fabric NoC adapters, (b) Relative performance comparison of the baseline NPU on Stratix 10 NX and the re-designed NPU on the 3 RADs we study, and (c) Performance of bigger workloads that can fit persistently in the larger on-chip memory resources of RAD2 and RAD3.**

compared to the 3 × 3 LAB region used in [50], as we are using 128-bit wide links vs. the 32-bit wide links of [50]. Table 3 shows the FPGA resource utilization of the NPU portions implemented on the FPGA fabrics of each of the three RADs and Fig. 12(a) shows the Quartus chip planner view of the FPGA portion implementation of the RAD3 instance.

We also verify that the matrix-vector multiplication units we chose to implement as hard accelerator blocks fit in the available ASIC sector area footprint. Since the silicon area footprint of Stratix 10 FPGA resources are proprietary information, we estimate the relative area as follows. First, we implement the matrix-vector multiplication unit on the Stratix 10 NX programmable fabric and convert its resource utilization results into equivalent ALMs [51] to get an area footprint breakdown of different components of the circuit. The matrix-vector multiplication unit utilizes the equivalent of 2.8 FPGA sectors with 68%, 21% and 11% of its area dedicated to BRAMs, ALMs and TBs, respectively. Prior studies show that ALMs, BRAMs, and DSPs/TBs have 26×, 3×, and 1.35× smaller area footprint when implemented in an ASIC without any reconfigurability or interfaces to the programmable routing [52], [53]. Therefore, we use a weighted average of these FPGA-to-ASIC area scaling ratios based on our circuit composition to get an approximate ASIC area footprint. The hard matrix-vector unit consumes less than 55% and 40% of the available ASIC sector area for the 4-thread (RAD2) and 1-thread (RAD3) variations respectively, leaving more than enough area for the NoC routers, adapters, links, and any additional hardened functionality. In the future, the RAD-Gen component of our flow, described in Sec. III-A, will automate any manual steps needed to obtain the FPGA results and will push the RTL implementation of the hard accelerator blocks through the ASIC design flow to obtain exact area and timing results.

C. PERFORMANCE RESULTS

Fig. 12(b) shows the relative performance comparison between the baseline latency-sensitive NPU on Stratix 10 NX from [43] and the re-designed NPU mapped to the three RAD instances we use in our study. Although RAD1 uses two FPGA fabrics, it does not benefit from any increase in the MVU compute resources compared to the baseline NPU. It only uses the resources of the second FPGA to add support for 4 interleaved thread executions. With the overhead of LI re-design and higher-latency NoC communication, RAD1 can achieve only 12% better performance on average compared to the baseline NPU. In comparison, the single-die RAD2 achieves 1.2× (1.32×) the performance of RAD1 (the baseline NPU) by exploiting the hardened MVU slices in the side coarse-grained accelerator blocks and interleaving four thread executions. Finally, the base die of RAD3 can implement the MVU slices of 4 NPU instances and frees the FPGA resources to implement the rest of their vector EW, LD and instruction dispatch units. This results in a significant 2.6× increase in average performance compared to the baseline NPU on a same form-factor FPGA without 3D stacking. In addition, since the hard matrix-vector multiplication units in RAD2 and RAD3 are designed to have bigger RFs, they can both run a new set of bigger workloads, shown in Fig. 12(c), that can not fit in the on-chip memory of the baseline NPU. These results show that RAD3 can achieve performance up to 145 TOPS on the LSTM-1536 workload.

Besides its ability to model a variety RAD architectures, RAD-Sim also enables us to study the effect of different architecture parameters on the performance of application designs. As an example, Fig. 13 shows the impact of changing the VC buffer size in the NoC routers of RAD2 for some of the NPU workloads (other workloads show the exact same trend but were omitted for brevity). Increasing the VC buffer size increases the silicon area footprint of the NoC routers, but acts as a bigger distributed storage for the packets traversing the NoC which can help avoid frequent NoC back pressures and decrease the overall communication latency. The results show that for the NPU traffic patterns over the NoC, VC buffer depths less than 8 flits can throttle performance, while increasing them beyond 8 flits yields little or no additional performance benefit.

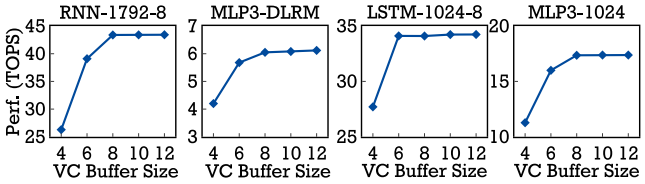


FIGURE 13. Effect of changing NoC VC buffer size on the end-to-end performance of some NPU workloads. Performance plateaus with no additional benefit for VC buffers that are more than 8 flits deep for all workloads.

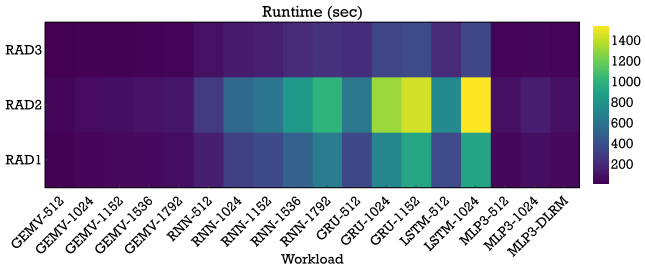


FIGURE 14. RAD-Sim runtime for simulating different NPU workloads on different RAD instances.

D. RUNTIME RESULTS

Fig. 14 shows a heatmap of RAD-Sim's runtime for simulating different workloads on the three example RADs we experiment with. It shows that runtime varies mainly depending on the number of simulation cycles for the different workloads. For example, runtime increases as workload size increases (e.g. RNN-512 vs. RNN-1792) and as workload complexity increases (e.g. RNN-1024 vs. LSTM-1024). Runtime also varies depending on the size of the simulated design (i.e. number of user design modules and accelerator blocks) and its NoC traffic patterns. Fig. 14 shows that RAD2 simulation runtime is generally higher than that of RAD1 since the RAD2 design is more distributed with modules connected to 54 NoC routers compared to 33 routers in RAD1.

On the other hand, RAD3 has a lower runtime since base die accelerator blocks are communicating with user modules connected to the same routers on the top FPGA die, and thus has simpler NoC traffic patterns than RAD2. In addition, although RAD3 has enough resources to implement 4 NPUs on the same device, it is enough to simulate only one instance since all four instances are completely independent with access to different sub-grids on the NoC (using different routers and links with XY dimension order routing). Across all our simulation runs, RAD-Sim's runtime ranges from 12 seconds to 34 minutes depending on the workload and RAD instance simulated. Evaluating a given RAD instance by simulating all NPU workloads takes between 1 hour and 3.5 hours, which can be further reduced by running different workloads in parallel (between 8 and 34 minutes if fully parallelized).

VI. CONCLUSION

Recent large-scale deployments of FPGAs in datacenters were mainly motivated by their faster time-to-solution compared to custom ASICs and their diverse high-bandwidth I/O interfaces that allow them to accelerate key datacenter functionalities on-the-fly at the data crossroads between different server end points. Building on these strengths, we have started to witness the emergence of novel RADs that combine the hardware flexibility of FPGAs, the high performance of domain-specialized accelerators, and the efficiency of packet-switched NoCs for system-level communication. In addition, advances in 3D chip fabrication and integration technologies are unlocking a whole new design space of multi-die RADs. However, RAD architects lack the tools to rapidly explore this huge design space and evaluate the effect of their design choices on end-to-end application performance. To this end, we develop RAD-Sim, an application-driven architecture simulator for modeling and evaluating candidate RAD architectures. It also allows early co-optimization of key application designs migrated from conventional FPGAs and the architecture parameters of a proposed RAD. We showcase RAD-Sim through a case study that maps the state-of-the-art NPU DL inference overlay on different example RAD instances. RAD-Sim's telemetry and visualization features pinpoint bottlenecks in the NPU on RADs with embedded NoCs, which we address with a new bandwidth-driven design approach and by adding multi-threading to increase tolerance of NoC latency. Our study also demonstrates that 3D-stacked RADs can increase average performance by a 2.6× compared to current FPGAs and achieve up to 145 TOPS on key DL workloads. We open source both RAD-Sim and the NPU example design for the broader research community to leverage in driving further innovations in RAD architecture.

REFERENCES

[1] A. Boutros and V. Betz, "FPGA architecture: Principles and progression," *IEEE Circuits Syst. Mag.*, vol. 21, no. 2, pp. 4–29, May 2021.
[2] B. Gaide, D. Gaitonde, C. Ravishankar, and B. T. Xilinx, "Adaptive compute acceleration platform: Versal architecture," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2019, pp. 84–93.
[3] D. Ingerly, S. Amin, and L. Aryasomayajula, "Foveros: 3D integration and the use of face-to-face chip stacking for logic devices," in *IEDM Tech. Dig.*, 2019, pp. 1–4.
[4] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "VTR 8: high-performance CAD and customizable FPGA architecture modelling," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 2, pp. 1–55, Jun. 2020.
[5] A. Boutros, E. Nurvitadhi, and V. Betz, "RAD-Sim: Rapid architecture exploration for novel reconfigurable acceleration devices," in *Proc. Int. Conf. Field Program. Log. Appl. (FPL)*, 2022, pp. 438–444.
[6] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 13–24, Oct. 2014.
[7] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–13.
[8] Z. Zhao, H. Sadok, N. Atre, J. C. Hoe, V. Sekar, and J. Sherry, "Achieving 100 Gbps intrusion prevention on a single server," in *Proc. USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2020, pp. 1083–1100.

[9] R. Ma, E. Georganas, A. Heinecke, A. Boutros, and E. Nurvitadhi, "FPGA-based AI smart NICs for scalable distributed AI training systems," 2022, *arXiv:2204.10943*.

[10] N. S. Kim and P. Mehra, "Practical near-data processing to evolve memory and storage devices into mainstream heterogeneous computing systems," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–4.

[11] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A configurable cloud-scale DNN processor for real-time AI," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 1–14.

[12] S. Yazdanshenas and V. Betz, "Interconnect solutions for virtualized field-programmable gate arrays," *IEEE Access*, vol. 6, pp. 10497–10507, 2018.

[13] M. S. Abdelfattah, A. Bitar, and V. Betz, "Design and applications for embedded networks-on-chip on FPGAs," *IEEE Trans. Comput.*, vol. 66, no. 6, pp. 1008–1021, Jun. 2017.

[14] M. S. Abdelfattah and V. Betz, "The case for embedded networks on chip on field-programmable gate arrays," *IEEE Micro*, vol. 34, no. 1, pp. 80–89, Jan./Feb. 2014.

[15] I. Swarbrick, D. Gaitonde, S. Ahmad, B. Gaide, and Y. Arbel, "Network-on-chip programmable platform in versal Tm ACAP architecture," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2019, pp. 212–221.

[16] *Speedster7t Network on Chip User Guide (UG089)*, Achronix, Santa Clara, CA, USA, 2019.

[17] S. Velagapudi and M. Honman, "Addressing memory-bandwidth and compute-intensive challenges with Intel Agilex M-series FPGAs," Intel Corp., Tech. Rep. WP-01313-1.0, 2022.

[18] M. Langhammer, E. Nurvitadhi, B. Pasca, and S. Gribok, "Stratix 10 NX architecture and applications," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2021, pp. 57–67.

[19] *Speedster7t Machine Learning Processing Unit Guide (UG088)*, Achronix, Santa Clara, CA, USA, 2019.

[20] A. Arora, S. Mehta, V. Betz, and L. K. John, "Tensor slices to the rescue: Supercharging ML acceleration on FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2021, pp. 23–33. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3529650

[21] R. Mahajan, R. Sankman, N. Patel, D.-W. Kim, K. Aygun, Z. Qian, Y. Mekonnen, I. Salama, S. Sharan, D. Iyengar, and D. Mallik, "Embedded multi-die interconnect bridge (EMIB)—A high density, high bandwidth packaging interconnect," in *Proc. IEEE 66th Electron. Compon. Technol. Conf. (ECTC)*, May 2016, pp. 557–565.

[22] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Titan: Enabling large and complex benchmarks in academic CAD," in *Proc. 23rd Int. Conf. Field Program. Log. Appl.*, Sep. 2013, pp. 1–8.

[23] A. Arora, A. Boutros, D. Rauch, A. Rajen, A. Borda, S. A. Damghani, S. Mehta, S. Kate, P. Patel, K. B. Kent, V. Betz, and L. K. John, "Koios: A deep learning benchmark suite for FPGA architecture and CAD research," in *Proc. 31st Int. Conf. Field-Program. Log. Appl. (FPL)*, Aug. 2021, pp. 355–362.

[24] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2013, pp. 86–96.

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

[26] A. Gutierrez, B. M. Beckmann, A. Dutu, J. Gross, M. LeBeane, J. Kalamatianos, O. Kayiran, M. Poremba, B. Potter, S. Puthoor, M. D. Sinclair, M. Wyse, J. Yin, X. Zhang, A. Jain, and T. Rogers, "Lost in abstraction: Pitfalls of analyzing GPUs at the intermediate language level," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 608–619.

[27] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked DRAM caches," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2011, pp. 454–464.

[28] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. Dreslinski, T. F. Wenisch, and S. Mahlke, "Composite cores: Pushing heterogeneity into a core," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2012, pp. 317–328.

[29] V. Seshadri, D. Lee, T. Mullins, and H. Hassan, "Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proc. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2017, pp. 273–287.

[30] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, 2011, pp. 1–12.

[31] S. Kanev, G.-Y. Wei, and D. Brooks, "XIOSim: Power-performance modeling of mobile X86 cores," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design ISLPED*, Jul. 2012, pp. 267–272.

[32] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated GPU modeling," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020, pp. 473–486.

[33] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 1–13, 2016.

[34] S. Angizi, Z. He, A. Awad, and D. Fan, "MRIMA: An MRAM-based in-memory accelerator," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 5, pp. 1123–1136, May 2020.

[35] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "HyGCN: A GCN accelerator with hybrid architecture," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 15–29.

[36] 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 *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014, pp. 97–108.

[37] Y. S. Shao, S. L. Xi, V. Srinivasan, G.-Y. Wei, and D. Brooks, "Co-designing accelerators and SoC interfaces using gem5-aladdin," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[38] G. Krishnan, S. K. Mandal, M. Pannala, C. Chakrabarti, J.-S. Seo, U. Y. Ogras, and Y. Cao, "SIAM: Chiplet-based scalable in-memory acceleration with mesh for deep neural networks," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5, pp. 1–24, 2021.

[39] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen, "Open cell library in 15 nm FreePDK technology," in *Proc. Symp. Int. Symp. Phys. Design*, Mar. 2015, pp. 171–178.

[40] M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar, "OpenRAM: An open-source memory compiler," in *Proc. 35th Int. Conf. Comput.-Aided Design*, Nov. 2016, pp. 1–6.

[41] M. K. Papamichael and J. C. Hoe, "CONNECT: Re-examining conventional wisdom for designing NoCs in the context of FPGAs," in *Proc. Int. Symp. Field Program. Gate Arrays (FPGA)*, 2012, pp. 37–46.

[42] M. Langhammer, G. Baeckler, and S. Gribok, "Spiderweb—High performance FPGA NoC," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2020, pp. 115–118.

[43] A. Boutros, E. Nurvitadhi, R. Ma, S. Gribok, Z. Zhao, J. C. Hoe, V. Betz, and M. Langhammer, "Beyond peak performance: Comparing the real performance of AI-optimized FPGAs and GPUs," in *Proc. Int. Conf. Field-Program. Technol. (ICFPT)*, Dec. 2020, pp. 10–19.

[44] E. Nurvitadhi, D. Kwon, A. Jafari, A. Boutros, J. Sim, P. Tomson, H. Sumbul, G. Chen, P. Knag, R. Kumar, R. Krishnamurthy, S. Gribok, B. Pasca, M. Langhammer, D. Marr, and A. Dasu, "Why compete when you can work together: FPGA-ASIC integration for persistent RNNs," in *Proc. IEEE 27th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2019, pp. 199–207.

[45] I. Swarbrick, D. Gaitonde, S. Ahmad, B. Jayadev, J. Cuppett, A. Morshed, B. Gaide, and Y. Arbel, "Versal network-on-chip (NoC)," in *Proc. IEEE Symp. High-Perform. Interconnects (HOTI)*, Aug. 2019, pp. 13–17.

[46] E. Nasiri, J. Shaikh, A. Hahn Pereira, and V. Betz, "Multiple dice working as one: CAD flows and routing architectures for silicon interposer FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 5, pp. 1821–1834, May 2016.

[47] A. Boutros, E. Nurvitadhi, and V. Betz, "Specializing for efficiency: Customizing AI inference processors on FPGAs," in *Proc. Int. Conf. Microelectron. (ICM)*, Dec. 2021, pp. 62–65.

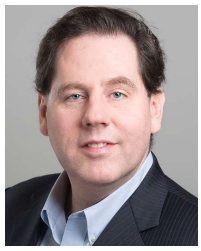
[48] M. S. Abdelfattah, A. Bitar, and V. Betz, "Take the highway: Design for embedded NoCs on FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2015, pp. 98–107.

[49] D. U. Becker, "Efficient microarchitecture for network-on-chip routers," Ph.D. Thesis, Dept. Elect. Eng., Stanford Univ., Stanford, CA, USA, 2012.

[50] M. S. Abdelfattah and V. Betz, "Design tradeoffs for hard and soft FPGA-based networks-on-chip," in *Proc. Int. Conf. Field-Programmable Technol.*, Dec. 2012, pp. 95–103.
[51] H. Wong, V. Betz, and J. Rose, "Comparing FPGA vs. Custom CMOS and the impact on processor microarchitecture," in *Proc. 19th ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2011, pp. 5–14.
[52] A. Boutros, S. Yazdanshenas, and V. Betz, "You cannot improve what you do not measure: FPGA vs. ASIC efficiency gaps for convolutional neural network inference," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 1–23, Sep. 2018.
[53] A. Boutros, S. Yazdanshenas, and V. Betz, "Embracing diversity: Enhanced DSP blocks for low-precision deep learning on FPGAs," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2018, pp. 35–357.

ANDREW BOUTROS (Student Member, IEEE) received the B.Sc. degree in electronics engineering from German University in Cairo, in 2016, and the M.A.Sc. degree in electrical and computer engineering from the University of Toronto, in 2018, where he is currently pursuing the Ph.D. degree. He was a Research Scientist at the Intel's Accelerator Architecture Laboratory, before he returned to the University of Toronto. His research interests include FPGA architecture and CAD, deep learning acceleration, and domain-specific architectures. He is an affiliate of the Intel/VMware Crossroads 3D-FPGA Academic Research Center, Vector Institute for Artificial Intelligence, and Center for Spatial Computational Learning. He received three best paper awards at Reconfig 2016, FPL 2018, and ICM 2021.

ERIKO NURVITADHI (Member, IEEE) received the M.S. and M.B.A. degrees from Oregon State University, in 2004, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, in 2010. He is currently a Principal Engineer at Intel. His research interests include computer architecture, hardware accelerators (FPGAs and ASICs), and their ecosystems (systems and software flows) for emerging workloads, such as AI, graphs, and analytics. He has over 60 peer-reviewed publications, 30 patents granted, and 50 patents pending. In 2020, he was recognized as one of Intel's top 30 inventors by Intel Patent Group and a Mahboob Khan Outstanding Liaison Award by Semiconductor Research Corporation. He has served on Program Committees for IEEE/ACM Conferences, such as DAC, FCCM, and FPGA.



VAUGHN BETZ (Fellow, IEEE) received the B.Sc. degree in electrical engineering from the University of Manitoba, in 1991, the M.S. degree in electrical and computer engineering from the University of Illinois at Urbana–Champaign, in 1993, and the Ph.D. degree in electrical and computer engineering from the University of Toronto, in 1998. He is the Original Developer of the widely used VPR FPGA placement, routing and architecture evaluation CAD flow, and a Lead Developer in the VTR project that has built upon VPR. He has co-founded Right Track CAD to commercialize VPR, and joined Altera upon its acquisition of Right Track CAD. He spent 11 years at Altera, ultimately as the Senior Director of Software Engineering, and is one of the architects of the Quartus CAD system and the first five generations of the Stratix and Cyclone FPGA families. He is currently a Professor at the University of Toronto. He holds 102 U.S. patents and has published over 100 technical articles in the FPGA area, 14 of which have won best or most significant paper awards. He is a fellow of the NAI and the EIC, and a Faculty Affiliate of the Vector Institute for Artificial Intelligence.