A Whole New World: How to Architect Beyond-FPGA Reconfigurable Acceleration Devices?

Andrew Boutros, Stephen More and Vaughn Betz
Department of Electrical and Computer Engineering, University of Toronto, Canada
E-mails: {andrew.boutros, stephen.more}@mail.utoronto.ca, vaughn@eecg.utoronto.ca

***Abstract*—Field-programmable gate arrays (FPGAs) have evolved beyond a fabric of soft logic and hard blocks surrounded by programmable routing to also incorporate high-performance networks-on-chip (NoCs), general-purpose processor cores and application-specific accelerators. These new reconfigurable acceleration devices (RADs) open up a myriad of architecture research questions, but require new computer-aided design tools for quantitative evaluation. In this work, we first enhance an existing RAD architecture simulator, RAD-Sim, to model high-bandwidth memory (HBM) and conventional DDR interfaces. We also introduce RAD-Gen which evaluates the silicon area and performance of the different components of a candidate RAD. We showcase the complete flow through a case study on accelerating deep learning recommendation models (DLRMs). Using RAD-Sim and RAD-Gen, we compare traditional FPGAs to RADs that incorporate hard NoCs and matrix-vector multiplication accelerators. This study demonstrates the utility of these tools in evaluating both the performance and implementation feasibility of different combinations of NoC and accelerator architecture parameters. The resulting RAD achieves an order of magnitude improvement in DLRM inference throughput and latency compared to prior FPGA implementations.**

I. Introduction

For several decades, field-programmable gate arrays (FPGAs) have continuously increased in capacity and heterogeneity. This has enabled the implementation of large customized hardware systems with higher efficiency than general-purpose processors and faster time-to-solution than application-specific integrated circuits (ASICs) in many areas [1], [2]. However, the continued growth of FPGAs faces two key challenges. First, as FPGAs and the designs implemented on them get bigger and external interfaces get faster, it becomes more challenging for the *soft* routing fabric to achieve the tremendous on-chip bandwidth required. This significantly impacts designer productivity as many optimization iterations are needed to close timing. Secondly, the FPGA computer-aided design (CAD) tools are having difficulty keeping compile times of ever-larger designs and FPGAs reasonable, further exacerbating the designer productivity problem. To face these challenges, new beyond-FPGA reconfigurable acceleration devices (RADs) are emerging. An exemplar of such devices is the Xilinx Versal architecture [3]. These devices combine the flexibility of an FPGA fabric, the generality of von-Neumann CPU cores, and the efficiency of application-specific accelerator cores. These diverse components are connected with a packet-switched network-on-chip (NoC). RADs mitigate the CAD scalability problem by realizing system-level communication with a NoC and moving some computation to coarse-grained accelerators. This reduces the size of the placement and routing problem and simplifies timing closure.

From an architect's perspective, designing such RADs is an arduous task as there are not only a large number of design choices for the FPGA fabric, NoC, and accelerators independently, but also complex interactions between these components. Crucially, for these architectures, applications are no longer just circuit netlists that can be used to evaluate a candidate architecture based on *application-agnostic* metrics such as operating frequency or routed wirelength, as in conventional FPGA architecture exploration. A RAD application can have some portions running on software-programmable accelerators and other parts on FPGA datapaths, interacting over an NoC that can suffer from application-dependent problems such as deadlock. Finally, there are not enough applications already designed for RADs to guide the architecture exploration, and the existing FPGA-based applications can not always be easily migrated to a RAD [4]. Therefore, a RAD architect needs to co-design both the architecture and the applications.

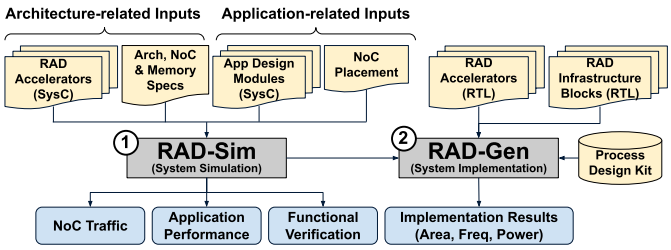

Fig. 1: RAD architecture exploration and evaluation flow.

Our work addresses this gap by providing tools for architects to explore this *whole new world* of possibilities when building novel RADs. We build on top of the open-source RAD-Sim tool [5] to complete the RAD architecture exploration flow illustrated in Fig. 1. The RAD-Sim component of the flow enables rapidly simulating application designs on RADs that incorporate FPGA fabrics, NoCs and specialized accelerators. This allows experimenting with different architecture specifications (i.e. knobs) and capturing their effect on end-to-end application performance. We enhance RAD-Sim by integrating a full DRAM simulator, DRAMsim3 [6], to enable modeling the cycle behavior and functionality of RADs with external memory sub-systems such as high-bandwidth (HBM) and double data rate (DDR) memories. We then introduce the second component of the flow, RAD-Gen, which can evaluate the ASIC implementation area and performance of the *hard* components in a RAD (e.g. NoC, accelerator cores) using the ASAP 7nm predictive process design kit (PDK) [7]. Together RAD-Sim and RAD-Gen can explore performance-cost trade-offs for a wide variety of candidate architectures. Finally, we showcase the flow and insights it can provide through a case study on accelerating deep learning recommendation model (DLRM) inference, which is a key datacenter workload. Our contributions include:

- Enhancing RAD-Sim to model devices with a variety of external memory subsystems (e.g. HBM, DDR).
- Introducing the RAD-Gen tool to evaluate the implementation cost and feasibility of hard RAD components.
- Presenting a case study to showcase the toolflow with DLRM inference acceleration as a key driving workload.

II. Background & Related Work

Embedded Hard NoCs in FPGAs: A large body of work has evaluated the use of packet-switched NoCs as a system-level interconnect solution for FPGAs. Some studies demonstrate the benefits of using the programmable logic and routing of the FPGA fabric to construct *soft* NoCs and ease timing closure in large FPGA

systems [8]–[10]. Other studies evaluated hardening a NoC in the FPGA fabric. In this case, the NoC routers are embedded as hard blocks in the fabric array, similar to digital signal processing blocks (DSPs) and block memories (BRAMs), with direct dedicated links between them. Hard NoCs were shown to be an order of magnitude more area-efficient and faster than their soft counterparts [11]. More recently, different variations of hard NoCs have been incorporated in commercial FPGAs [12]–[14]. We believe hard NoCs will remain a fundamental component in future RADs to efficiently interconnect all the different system components, and therefore they are a key focus of our RAD architecture exploration.

RAD-Sim: As shown in Fig. 1, RAD-Sim takes as inputs architecture and NoC specifications, SystemC models of any hard accelerators and application design modules that reside on the programmable fabric, and location assignments of different design modules to NoC router adapters. RAD-Sim internally uses Booksim [15] for NoC performance modeling, but augments it to model the functionality of NoC communication to enable verification of entire application designs. In addition, it (1) provides simulation infrastructure components such as AXI NoC adapters, (2) automates interfacing between modules and the NoC adapters at specified locations, and (3) includes telemetry utilities for recording and visualizing execution traces. RAD-Sim enables rapid *what-if* analysis of different design choices and evaluates the effect of these choices on end-to-end application performance. Previously, RAD-Sim did not model components outside the RAD chip, limiting the type of applications that could be evaluated. In this work, we address this limitation by extending RAD-Sim to model the performance and functionality of RADs with external memory subsystems.

III. DRAM-Enhanced RAD-Sim

Memory systems are a key focus in computer architecture research and there are several open-source cycle-accurate DRAM simulators to facilitate their exploration [16]–[18]. We integrate DRAMsim3 [6] into RAD-Sim to enable modeling of RADs with external memories. DRAMsim is written in C++ and ships with over 80 configuration files that model the timing and features of a wide variety of modern DRAM protocols (e.g. DDR4, GDDR6, HBM). It has also been integrated in multiple CPU-system simulators such as Gem5 [19].

For our use case in RAD-Sim, we need to model scenarios in which a design module (mapped to either the FPGA fabric or a hard accelerator core) reads/writes data from/to an external memory over the NoC. Therefore, we build a RAD-Sim-compatible SystemC wrapper around DRAMsim which presents a standard AXI memory-mapped (AXI-MM) interface to connect to the RAD-Sim NoC on one side, and interfaces with DRAMsim on the other side. We also add data book-keeping in our wrapper since RAD-Sim simulates not only performance but also functionality, which requires sending back the correct memory contents in a read response. This wrapper module runs at the memory controller clock frequency, receives AXI-MM transactions, and translates them to DRAMsim memory requests. It also implements custom read/write callback functions that are invoked by DRAMsim when a memory request is serviced. These callback functions prepare an AXI-MM response transaction with the memory contents at the requested address to be sent out over the NoC. Our DRAMsim wrapper can also optionally implement reordering logic to return responses in the order the requests were received.

We simulate an 8-channel HBM configuration in RAD-Sim and compare the results to both RTL simulations and hardware measurements of a similar HBM configuration on a Stratix 10 NX FPGA. The comparison shows similar trends for bandwidth and latency, with DRAMsim3 being $\sim$20% pessimistic for burst-1 accesses which will add some conservatism to our case study detailed in Section V. Our simulation results also agree with the HBM characterization results in [20].

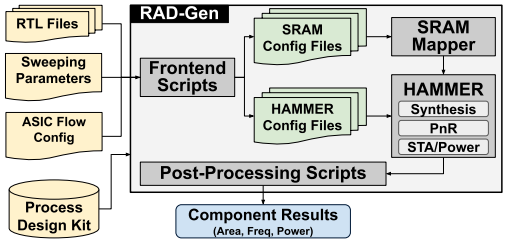

Fig. 2: Overview of the RAD-Gen tool flow.

IV. RAD-Gen

RAD-Sim only evaluates application performance on candidate RAD architectures, but does not address the silicon cost (or even feasibility) of their implementation. For example, a specific hard accelerator core might provide huge cycle-count performance benefits but might not fit within the silicon area budget or run at a slower frequency than what RAD-Sim assumed. For this reason, we introduce RAD-Gen, the second key component of the RAD architecture exploration and evaluation flow in Fig. 1. After rapidly iterating in RAD-Sim to narrow down the set of candidate RAD architectures, RAD-Gen can be used to perform a more detailed analysis of these options by evaluating the implementation area, speed and power consumption of their key components. Fig. 2 shows the internals of RAD-Gen. The inputs to the tool are parameterizable RTL descriptions of common RAD infrastructure blocks (e.g. NoC routers) and/or candidate accelerator cores, a list of RTL parameter values to sweep, and ASIC flow configurations such as target frequencies, placement constraints, and target standard cell density.

A. SRAM Mapper

Many RAD components contain embedded SRAM memories of various sizes; RAD-Gen extracts all the SRAM configurations (width, depth and number of ports) from its inputs and then builds appropriate SRAM blocks. If an SRAM compiler is available for the target process, it can be invoked to evaluate the area and delay of each configuration. However, in many cases (including for the ASAP7 PDK we target) an SRAM compiler is not available so instead RAD-Gen invokes a built-in *SRAM mapper*. The SRAM mapper automatically composes the required configurations using the SRAM macros available and external stitching logic. This results in a sub-optimal implementation compared to a single SRAM bank, but provides an upper bound on SRAM area in cases where an SRAM compiler is not readily available.

B. Reusable Physical Design Flow using HAMMER

To run the ASIC physical design tools, RAD-Gen uses HAMMER [21] from the UC Berkeley Chipyard framework [22]. It is a Python tool that provides a unified syntax to enable design flow reusability across different tools, vendors and process technologies. It already ships with multiple backend plug-ins for most of the commonly used Cadence, Synopsys and Siemens tools for synthesis, place and route (PnR), and static timing analysis (STA). The front-end RAD-Gen scripts use the provided inputs to generate project directories with their corresponding HAMMER `.yaml` configuration files for the different parameter variations specified by the user. Then it launches these runs in parallel based on a user-specified CPU core allocation for each run.

C. Post Processing Scripts

After Synthesis, PnR and STA complete, a set of post-processing scripts parse the results and generate the final timing, area and power reports for the different standalone RAD blocks/components. In some cases, additional post-processing is also required. For example, the ASAP 7nm PDK scales the library LEFs and QRC techfiles by a factor of 16$\times$ to work around special license requirements for sub-20nm processes in Cadence Innovus [23]. The HAMMER plug-in

for ASAP7 already provides a script that scales the output GDS file back to 7nm, and our post-processing scripts automatically stream the resulting scaled GDS file in the Cadence Virtuoso command line interface, and extract its 7nm bounding box area to present the final area results to the user. The extracted bounding box area provides a conservative estimate when performing block-level analysis since in a flat PnR some of the empty spaces inside the bounding box could potentially be used by the physical design tools.

V. Case Study for RAD Architecture Exploration

In this section, we present a full example case study to illustrate how RAD-Sim and RAD-Gen can be used together to perform application-driven architecture exploration for RADs. The process typically starts with selecting key target workloads that represent application domains of interest to drive the architecture exploration of a new device. For our case study, we chose DLRM inference acceleration as it is one of the major workloads in the deep learning domain and is part of the MLPerf benchmark suite [24].

A. Deep Learning Recommendation Models (DLRMs)

DLRMs are a key workload for many online service providers such as Facebook, Amazon, Alibaba, Netflix, and Google. They are ubiquitously used to personalize online retail [25] and predict click-through rates for advertisements, news feeds, and search results [26]–[28], having direct impact on the revenue of these companies. According to [29], DLRMs are the single biggest deep learning workload in Facebook datacenters. There are different variations of DLRMs, but Fig. 3a shows their general architecture. The inputs to DLRMs are dense features that represent numerical data (e.g. user age) and a set of sparse one/multi-hot-encoded categorical information (e.g. country, previously liked products). The dense features are optionally processed using a relatively small multi-layer perceptron (MLP), commonly referred to as the *bottom MLP*. On the other hand, N sparse feature vectors are treated as indicies for looking up one or multiple dense vectors (followed by a pooling operation) from each of the N corresponding embedding tables (ETs). The embedding layer transforms the categorical inputs from a D-dimensional sparse space into a d-dimensional dense space where $D >> d$. Then, a feature interaction stage combines the dense embedding vectors and the raw or processed dense features using concatenation, weighted sum or element-wise multiplication, and then feeds the resulting vector into the *top MLP* to compute the final model prediction. Variations of DLRMs can differ in whether they include a bottom MLP or not, the size of the MLPs, the feature interaction operation, the number and sizes of embedding tables, and the number of lookups per table. For our case study, we use two production models from Alibaba which were also used in [30]. Both models omit the bottom MLP, perform a single lookup per embedding table, and have a 3-layer top MLP with hidden dimensions of 1024, 512, and 256 neurons. The small and large models have 47 and 98 embedding tables with a memory footprint of 675 MB and 7.03 GB respectively, when using half-precision floating point (`fp16`) numerical format.

B. DLRM Accelerator Architecture

First, we design a DLRM inference accelerator architecture for conventional commodity FPGAs as depicted in Fig. 3b. The embedding lookup module receives the input features (from a host CPU or network interface) and issues memory read requests for the different embedding tables. For simplicity, we store all embedding tables in external memories. We interface with two DDR4 channels and two stacks of HBM similar to what is available in modern FPGA devices such as the Xilinx Alveo U280 or Intel Stratix 10 MX FPGAs [31]. Accessing as many embedding tables as possible in parallel is key for achieving low DLRM inference latency. The HBM in modern FPGA devices can operate in pseudo-channel mode in which each of the 8 HBM channels can be split into

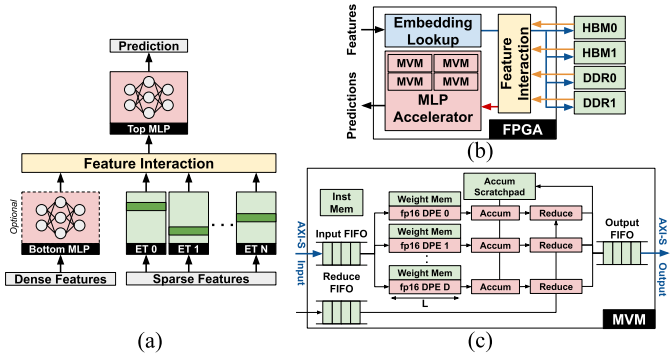

Fig. 3: (a) Typical DLRM organization, and block diagrams of the (b) DLRM accelerator and (c) MVMs used for accelerating the MLPs.

TABLE I: Resources utilization breakdown for the DLRM accelerator on a conventional FPGA.

Module	ALMs	DSPs	BRAMs
Memory Interfaces	22,949	-	-
Embedding Lookup	769	-	-
Feature Interaction	2,329	-	100
MVMs & Collector	89,906	3,264	1,420
Total	**115,953 (17%)**	**3,264 (82%)**	**1,520 (22%)**

two half-width pseudo channels, providing 16 parallel lookups per stack. However, a limitation of DRAMsim3 (which we integrate in RAD-Sim) is that it does not support pseudo-channel modeling. Therefore, we use the HBMs in their legacy 8-channel mode which offers the same bandwidth as pseudo-channel mode but only 8 parallel full-width lookups per stack. The memory read responses are sent back to the feature interaction module which implements a per-model customized crossbar and control finite-state machine (FSM) to select and concatenate the valid bytes of each 64B memory response (since embedding table entries have different sizes ranging from 4 to 32 elements in the models we study). Then, the concatenated vector is passed to the MLP accelerator to compute the final prediction. For the MLP acceleration, we implement a streaming-style matrix-vector multiplication (MVM) engine as shown in Fig. 3c. It consists of D `fp16` dot-product engines (DPEs), each with L multiplication lanes. Each DPE is tightly coupled with a persistent weight memory which stores the MLP weight matrices. The incoming input vectors are broadcast from the input FIFO to all the DPEs to be multiplied by different rows of the matrix. This is followed by accumulators and a scratchpad memory to accumulate results over time, and a reduction unit to reduce the final accumulation outputs with those produced by another MVM engine if needed. This MVM engine is orchestrated by instructions stored in a small 64KB instruction memory, and has AXI-Streaming (AXI-S) interfaces for input and output vectors with direct MVM-to-MVM connections for inter-MVM reduction.

C. Results on a Conventional FPGA

We implement the embedding lookup, feature interaction, and MVM modules in SystemVerilog and compile the full design for the largest and fastest speed grade Intel Stratix 10 MX 2100 FPGA using Quartus Prime Pro 22.4 to obtain the operating frequency and resource utilization results[1]. Limited by DSP resources, we were able to fit three MVMs with 32 DPEs each and 32 lanes per DPE ($D = L = 32$) for the MLP acceleration portion. Table I shows the resource utilization breakdown, and Fig. 4 shows the chip planner view of the design placement and routing congestion heatmap. The

[1]At the time of writing, newer-generation Agilex devices with HBMs are still not supported in the latest available Quartus version. Therefore, we use a Stratix 10 device to capture a realistic FPGA placement and routing but scale the area results to 7nm when we calculate the LAB-equivalent area of NoC routers and accelerator core silicon area budget in Section V-D.

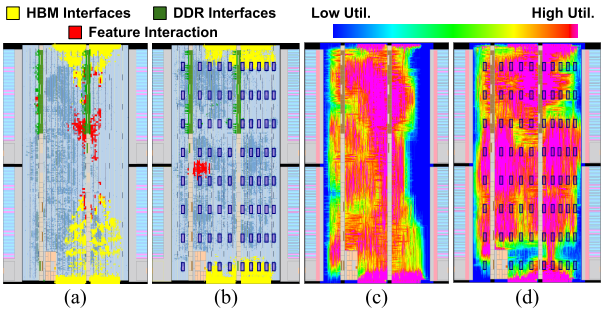

Fig. 4: Chip planner view for the placement and routing heatmap of the DLRM inference accelerator on a conventional FPGA (a & c) and hard-NoC-enhanced FPGA (b & d).

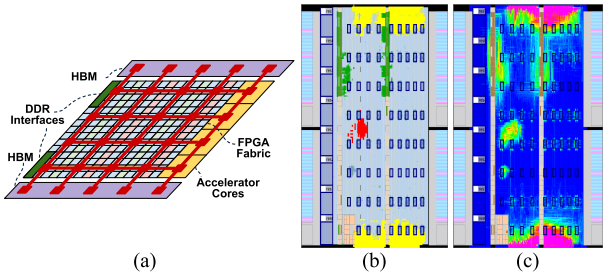

Fig. 5: (a) Example RAD for architecture exploration case study, and (b) module placement and (c) routing heatmap of the fabric portion of the DLRM accelerator on the RAD. The leftmost column of sectors is reserved for ASIC accelerator cores.

HBM interfaces in the Stratix 10 MX device (highlighted in yellow in Fig. 4a) are at the top and bottom of the device and the DDR interfaces (green) use the I/O columns within the fabric. Therefore, the feature interaction module (red) talking to the external memories is stretched between these interfaces at opposite sides of the chip, resulting in significantly long critical path delays. This, along with the severe routing congestion caused by all the wide buses from/to the memory interfaces and within the design modules (Fig. 4c), causes the design to operate at 80 MHz.

We also implement a SystemC version of each of these modules to be used in RAD-Sim. When simulating the end-to-end baseline design in our DRAM-enhanced RAD-Sim, it achieves $139K$ inferences per second at $12.5\mu s$ inference latency, and $82K$ inferences per second at $19\mu s$ inference latency for the small and large Alibaba models, respectively. These results show 31% and 17% lower latency but $0.46\times$ and $0.42\times$ the throughput of the state-of-the-art `fp16` results in [30] for the small and large models, respectively. Considering that the accelerator from [30] runs at $1.5\times$ higher frequency, uses $1.4\times$ more DSPs, and redesigns the data structure of embedding tables to reduce the number of memory lookup rounds, these results give us confidence that our baseline design on a conventional FPGA is representative of state-of-the-art FPGA-based DLRM accelerators.

D. Area Budgets for Hard NoC and Accelerator Cores

In this case study, we want to explore RAD architectures similar to that illustrated in Fig. 5a. These RADs incorporate a side complex of hard accelerator cores for common functionalities and a hard NoC that is the only way for modules implemented on the programmable fabric to access external memories (similar to the Xilinx Versal architecture [12]) and/or communicate with the hard accelerator cores. To realize such RADs, some of the programmable fabric logic blocks (LBs) would be replaced by NoC routers (a new type of hard block), and some of the programmable fabric sectors would be completely replaced by ASIC accelerator cores. Therefore, we need to estimate the sector area to determine an area budget for our accelerator cores, as well as the LB area to quantify the FPGA capacity that would be sacrificed to embed NoC routers with a specific configuration.

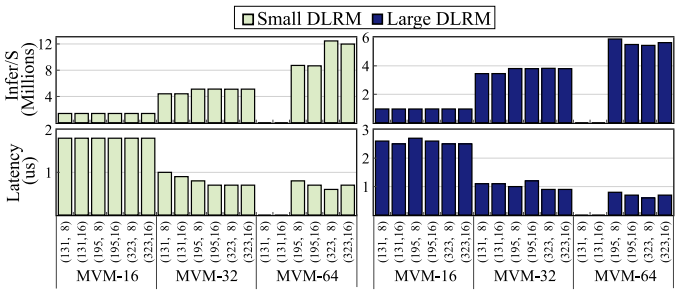

Fig. 6: DLRM acceleration throughput (upper half) and latency (lower half) for the two Alibaba models on RADs with different parameters. The values in brackets are (NoC links witdth, VC buffer size).

Since this data is proprietary, we perform a best-guess analysis based on publicly disclosed information. Based on [32], the fabric die area of a 2.8M logic element Stratix 10 device is $560mm^2$. Since this device consists of 12×9 fabric sectors, a single sector area is $5.2mm^2$ at the 14nm process. Then, we scale this area by a factor of $3\times$ (derived from ASAP7 results for various designs before and after GDS scaling) to obtain a $1.7mm^2$ area budget for each accelerator core at 7nm. We reserve a column of eight fabric sectors in the Stratix 10 MX 2100 device we use in our experiments (as shown in Fig. 5b) to be replaced by eight ASIC accelerator cores. Also, we know that the relative areas of Stratix-V BRAMs and DSPs (which did not change significantly for Stratix 10) to adaptive logic modules (ALMs) are $40\times$ and $30\times$ respectively [33], and that IOs typically consume around 20% of the FPGA die area in large devices [34]. Using these numbers along with the resource composition of the Stratix 10 GX2800 device, we can estimate that 50% of the $560mm^2$ is consumed by LBs. Therefore, the area of a single LB is roughly $1000\mu m^2$ when scaled to 7nm.

We perform an experiment to evaluate the effect of embedding a hard NoC on the placement and routing of our baseline DLRM accelerator design. We reserve an 8×9 mesh of exclusive logic-locked regions in the FPGA fabric to represent the hard NoC routers and their NoC-to-fabric adapters as shown in Fig. 4b. Then, instead of directly connecting the DLRM accelerator modules to each other, we connect them to black-box (empty) router modules and manually assign these router modules to their corresponding logic-locked regions to mimic an FPGA fabric with an embedded hard NoC. Fig. 4b shows the effect on the placement of modules; the (red) feature interaction module and (yellow) HBM interfaces are no longer stretched across the fabric as when connected using the programmable routing (Fig. 4a). Instead, they are more localized close to their access point NoC routers. However, when looking at the routing congestion heatmap of this scenario in Fig. 4d, we observe that the routing congestion was not improved. The main reason is that the three MVM modules are very coarse grained (each consumes 27% of the fabric DSPs) and very routing-intensive. The hard NoC provides efficient system-level interconnect between modules; however, designers sometimes still need to re-think the granularity of NoC-attached modules and their intra-module routing to implement high-performance application designs on RADs as detailed in [4].

E. DLRM Acceleration Performance on RADs

We first use RAD-Sim to simulate our DLRM inference accelerator when mapped to different variations of RAD architectures. We explore NoCs with different link widths of 131, 195, and 323 bits (which packetize an AXI transaction with 512-bit data into 8, 4, and 2 flits respectively), and different VC buffer sizes of 8 and 16 flits. We also explore MVM engines with different compute capabilities: $16\times$ 16-lane DPEs, $32\times$ 32-lane DPEs, and $64\times$ 64-lane DPEs (i.e. MVM-16, MVM-32, MVM-64). Although RAD-Sim allows

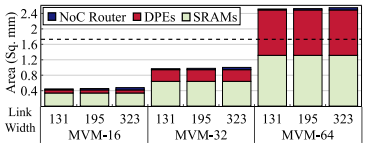

Fig. 7: RAD-Gen area results of accelerator cores with different MVM sizes and NoC link widths. Dashed line is the area budget for the sectors.

TABLE II: Logic block (LB) region reserved for a NoC router and % of LBs replaced by 8×8 mesh NoC.

NoC Links Width	Router Size (LBs)	Total % LBs
131	5×5	2.6%
195	6×6	3.7%
323	8×8	6.6%

experimenting with many more architecture aspects, we focus on just three key parameters for showcasing the tool. For simulations, we assume that the NoC, NoC adapters, MVM accelerator cores, and soft logic modules run at 1.5 GHz, 1.2 GHz, 1 GHz, and 300 MHz, respectively. Later when we perform RAD-Gen experiments, we verify that these frequencies are all achievable.

Fig. 6 presents the performance results for all RAD variations swept for both the small and large Alibaba DLRM models. In all experiments, changing the VC buffer size does not have any noticeable effect on performance. The results also show that for both models, the less compute capable MVM-16 engines are the bottleneck when used, resulting in no change in performance when varying the NoC-related parameters. Additionally, combining the narrowest NoC links with the highest number of MVM lanes (and thus the highest MVM input/output bitwidth) results in a large number of flits per AXI transaction causing severe congestion in the NoC and eventually leading to a deadlock. This highlights the utility of RAD-Sim in uncovering (and potentially solving) such application-dependent functionality problems for specific combinations of architectural parameters. For the small DLRM model, increasing the compute capability of the MVMs gradually shifts the bottleneck from the MLP compute to the data movement. This is manifested by the throughput improvement when increasing the width of the NoC links from 131 to 195 and from 195 to 323 in the MVM-32 and MVM-64 cases, respectively. The large model shows a similar trend with the NoC link width having less pronounced impact on performance. For latency results, a significant improvement is achieved by moving from MVM-16 to MVM-32, highlighting that the latency was dominated by the MLP compute in the MVM-16 case. However, only a slight latency improvement is achieved by further increasing the compute capability of the MVMs. These results show that the MVM-64 variation is desirable for higher throughput, but requires wider NoC links to be effective. On the other hand, medium-sized MVMs are enough to achieve most of the latency reduction. RAD-Sim enables efficient exploration of the architecture space – this entire parameter sweep takes less than an hour on an Intel Core i5-12600 CPU clocked at 3.7 GHz with 32 GB of RAM.

F. Physical Implementation Trade-offs

We use RAD-Gen to evaluate the implementation cost and feasibility of the different RAD variations. Since RAD-Sim already showed that VC buffer size does not affect performance, we only evaluate the more area-efficient NoC routers with VC buffers of 8 flits. We use RAD-Gen with the ASAP 7nm PDK [7], and configure it to use Cadence Genus v20.10 for synthesis, Cadence Innovus v21.12 for PnR, and Synopsys PrimeTime v2017.09 for STA. We set aggressive frequency targets of 1.75 GHz and 1.4 GHz for the NoC routers and MVM components respectively, and a target standard cell density of 70%. The tools reported that all the NoC router and MVM component variations we experiment with can meet timing at 1.6 GHz. Fig. 7 shows the silicon area breakdown of the ASIC accelerator sectors (i.e. MVM engine and the NoC router it is attached to) relative to the available area budget (dashed line) that we calculated in Section V-D. It shows that the MVM-64 design points are infeasible as they are $45-47\%$ bigger than the available budget. The MVM-16 and MVM-32 data points are all feasible and consume at most 28% and 58% of the available budget, leaving space to harden additional functionality for other applications or expand the SRAM capacity of these sectors (which can be used as a bigger but higher access latency alternative to the fabric BRAMs). Since the MVM-32 variations can achieve significant performance improvements compared to MVM-16 based on RAD-Sim results, they are a more reasonable design choice. The NoC router embedded within each MVM constitutes a small portion of the accelerator area ($< 6\%$). However, since most of the NoC routers are embedded in the FPGA fabric, their area impact is more significant in terms of the soft logic replaced. Table II shows the amount of soft logic replaced by embedding a mesh of 8×8 routers in the fabric for each NoC variation as shown in Fig. 5b. This shows that NoC routers with 131-bit or 195-bit wide links would replace 2.6% and 3.7% of the Stratix 10 MX fabric respectively for embedding the whole NoC (after excluding the resources replaced by the ASIC accelerator sectors). Moving to even wider 323-bit wide links, increases this cost to 6.6% but, when looking at RAD-Sim results, does not provide noticeable performance gains. Therefore, a NoC with 195-bit links in this case offers a better trade-off. Fig. 5c shows that moving the MVM functionality to the hard accelerator cores reduces routing congestion, and frees up the flexible FPGA fabric to implement any additional (usecase-specific) pre/post-processing stages.

TABLE III: DLRM performance of future RADs in context with CPU and state-of-the-art FPGA solutions (B: batch size).

		CPU (B=1)	CPU (B=2048)	MicroRec [30]	Candidate RAD
Small	Latency (ms)	3.34	28.18	0.0163	0.0008
	Infer/s ($\times 10^3$)	0.3	72.7	305	5,117
Large	Latency(ms)	7.48	56.98	0.0226	0.001
	Infer/s ($\times 10^3$)	0.13	35.9	195	3,809

G. The Potential of RADs for DLRM Acceleration

Although this is not the key focus of the paper, Table III aims to put the potential performance of the resulting candidate RAD from our architecture exploration in context with existing CPU and FPGA solutions for the Alibaba models used in our case study. We compare our (conservatively modeled 7nm) RAD with eight MVM-32 accelerator cores and an 8×9 NoC with 195-bit wide links to results from [30] for 16 vCPUs of an Intel Xeon E5-2686 @ 2.30GHz with AVX2 FMA (Intel 14nm) and a state-of-the-art FPGA-based DLRM inference accelerator on a Xilinx Alveo U280 (TSMC 16nm). This comparison shows that next-generation RADs can achieve an order of magnitude higher throughput and lower latency with up to 5.1M inferences per second.

VI. Conclusion

General-purpose processors no longer improve performance only by scaling up their core count, but also now often incorporate specialized accelerators. Similarly, FPGA capabilities will grow not just with ever-larger fine-grained programmable fabrics, but also with diverse on-die specialized accelerator cores. Architecture exploration of these beyond-FPGA reconfigurable devices requires new tooling to model the interactions between soft fabrics, hard NoCs, accelerator blocks and off-chip memory. To this end we have extended the RAD-Sim tool to simulate off-chip memory, and developed the RAD-Gen tool to enable area, performance and power estimates of RAD components. Through a case study on DLRM acceleration, we showed that RAD-Sim enables co-optimization of the NoC and accelerator cores, while RAD-Gen evaluates the implementation cost and feasibility of various design points.

Acknowledgements

The authors would like to thank the Intel/VMware Crossroads 3D-FPGA Academic Research Center for funding support.

REFERENCES

[1] A. Putnam *et al.*, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 13–24, 2014.

[2] A. Caulfield *et al.*, "A Cloud-Scale Acceleration Architecture," in *International Symposium on Microarchitecture (MICRO)*, 2016.

[3] B. Gaide *et al.*, "Xilinx Adaptive Compute Acceleration Platform: Versal Architecture," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019.

[4] A. Boutros, E. Nurvitadhi, and V. Betz, "Architecture and Application Co-Design for Beyond-FPGA Reconfigurable Acceleration Devices," *IEEE Access*, vol. 10, pp. 95 067–95 082, 2022.

[5] ——, "RAD-Sim: Rapid Architecture Exploration for Novel Reconfigurable Acceleration Devices," in *International Conference on Field-Programmable Logic and Applications (FPL)*, 2022.

[6] S. Li *et al.*, "DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.

[7] L. T. Clark *et al.*, "ASAP7: A 7-nm FinFET Predictive Process Design Kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[8] M. Langhammer *et al.*, "Spiderweb: High Performance FPGA NoC," in *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 115–118.

[9] N. Kapre and J. Gray, "Hoplite: Building Austere Overlay NoCs for FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–8.

[10] M. K. Papamichael and J. C. Hoe, "CONNECT: Re-Examining Conventional Wisdom for Designing NoCs in the Context of FPGAs," in *ACM/SIGDA international symposium on Field Programmable Gate Arrays (FPGA)*, 2012, pp. 37–46.

[11] 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, 2013.

[12] I. Swarbrick *et al.*, "Network-on-Chip Programmable Platform in Versal ACAP Architecture," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019, pp. 212–221.

[13] Achronix Corp., "Speedster7t Network on Chip User Guide (UG089)," 2019.

[14] S. Velagapudi and M. Honman, "Addressing Memory-Bandwidth and Compute-Intensive Challenges with Intel Agilex M-Series FPGAs (WP-01313-1.0)," 2022.

[15] N. Jiang and other, "A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013.

[16] Y. Kim *et al.*, "Ramulator: A Fast and Extensible DRAM Simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 2015.

[17] M. K. Jeong *et al.*, "DrSim: A Platform for Flexible DRAM System Research," http://lph.ece.utexas.edu/public/DrSim.

[18] N. Chatterjee *et al.*, "USIMM: The Utah Simulated Memory Module," *University of Utah, Technology Report*, 2012.

[19] N. Binkert *et al.*, "The gem5 Simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[20] H. Huang *et al.*, "Shuhai: A Tool for Benchmarking High Bandwidth Memory on FPGAs," *IEEE Transactions on Computers*, vol. 71, no. 5, pp. 1133–1144, 2021.

[21] H. Liew *et al.*, "HAMMER: A Modular and Reusable Physical Design Flow Tool," in *ACM/IEEE Design Automation Conference (DAC)*, 2022.

[22] A. Amid *et al.*, "Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[23] L. T. Clark *et al.*, "Design Flows and Collateral for the ASAP7 7nm FinFET Predictive Process Design Kit," in *International Conference on Microelectronic Systems Education (MSE)*, 2017.

[24] V. J. Reddi *et al.*, "MLPerf Inference Benchmark," in *International Symposium on Computer Architecture (ISCA)*, 2020.

[25] B. Smith and G. Linden, "Two Decades of Recommender Systems at Amazon.com," *IEEE Internet Computing*, vol. 21, no. 3, pp. 12–18, 2017.

[26] G. Zhou *et al.*, "Deep Interest Network for Click-Through Rate Prediction," in *ACM International Conference on Knowledge Discovery & Data Mining (KDD)*, 2018.

[27] M. Naumov *et al.*, "Deep Learning Recommendation Model for Personalization and Recommendation Systems," *arXiv:1906.00091*, 2019.

[28] H.-T. Cheng *et al.*, "Wide & Deep Learning for Recommender Systems," in *Workshop on Deep Learning for Recommender Systems (DLRS)*, 2016.

[29] D. Mudigere *et al.*, "Software-Hardware Co-Design for Fast and Scalable Training of Deep Learning Recommendation Models," in *International Symposium on Computer Architecture (ISCA)*, 2022.

[30] W. Jiang *et al.*, "MicroRec: Efficient Recommendation Inference by Hardware and Data Structure Solutions," *Proceedings of Machine Learning and Systems (MLSys)*, 2021.

[31] Y.-k. Choi *et al.*, "HBM Connect: High-Performance HLS Interconnect for FPGA HBM," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021.

[32] D. Greenhill *et al.*, "A 14nm 1GHz FPGA with 2.5D Transceiver Integration," in *International Solid-State Circuits Conference (ISSCC)*, 2017.

[33] R. Rashid *et al.*, "Comparing Performance, Productivity and Scalability of the TILT Overlay Processor to OpenCL HLS," in *International Conference on Field-Programmable Technology (FPT)*, 2014.

[34] V. Betz, "Grand Challenges in FPGA Research (Workshop Presentation)," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2007.