Koios: A Deep Learning Benchmark Suite for FPGA Architecture and CAD Research

Aman Arora[1], Andrew Boutros[2], Daniel Rauch[1], Aishwarya Rajen[1], Aatman Borda[1], Seyed Alireza Damghani[3], Samidh Mehta[1], Sangram Kate[1], Pragnesh Patel[1], Kenneth B. Kent[3], Vaughn Betz[2], Lizy K. John[1]
[1]*The University of Texas at Austin* [2]*University of Toronto & Vector Institute for AI* [3]*University of New Brunswick*
E-mail: aman.kbm@utexas.edu

***Abstract*—With the prevalence of deep learning (DL) in many applications, researchers are investigating different ways of optimizing FPGA architecture and CAD to achieve better quality-of-results (QoR) on DL-based workloads. In this optimization process, benchmark circuits are an essential component; the QoR achieved on a set of benchmarks is the main driver for architecture and CAD design choices. However, current academic benchmark suites are inadequate, as they do not capture any designs from the DL domain. This work presents a new suite of DL acceleration benchmark circuits for FPGA architecture and CAD research, called Koios. This suite of 19 circuits covers a wide variety of accelerated neural networks, design sizes, implementation styles, abstraction levels, and numerical precisions. These designs are larger, more data parallel, more heterogeneous, more deeply pipelined, and utilize more FPGA architectural features compared to existing open-source benchmarks. This enables researchers to pin-point architectural inefficiencies for this class of workloads and optimize CAD tools on more realistic benchmarks that stress the CAD algorithms in different ways. In this paper, we describe the designs in our benchmark suite, present results of running them through the Verilog-to-Routing (VTR) flow using a recent FPGA architecture model, and identify key insights from the resulting metrics. On average, our benchmarks have 3.7× more netlist primitives, 1.8× and 4.7× higher DSP and BRAM densities, and 1.7× higher frequency with 1.9× more near-critical paths compared to the widely-used VTR suite. Finally, we present two example case studies showing how architectural exploration for DL-optimized FPGAs can be performed using our new benchmark suite.**

I. Introduction

With compute and data intensive deep learning (DL) becoming a major component of many applications, specialized hardware acceleration of such workloads has become a commonplace. More recently, field-programmable gate arrays (FPGAs) have been shown to deliver state-of-the-art performance when accelerating different DL workloads because of their massive parallelism, flexibility and energy efficiency [1], [2]. With new DL use cases emerging faster than ever, FPGAs are also starting to adapt. This includes the emergence of DL-optimized FPGA fabrics [3], the integration of FPGAs with specialized DL accelerators [4], [5], and also tuning FPGA CAD tools to the properties of these workloads [6].

In general, the development of novel FPGA architectures and CAD algorithms depends mainly on a versatile framework that consists of three main components: (1) a set of benchmarks written in a hardware description language or synthesized using high-level synthesis, (2) an architecture model that captures the organization of FPGA blocks and routing architecture as well as area/timing/power models from circuit-level implementations, and (3) a CAD flow that synthesizes the given benchmarks then implements them on a given FPGA architecture [7]. Although most research efforts in the FPGA community are focused on architecture and CAD, benchmarks actually play a crucial role in this flow. The quality-of-results (QoR) achieved on a specific set of benchmarks is the main driver for architecture and CAD design choices. As a result, it is essential that these benchmarks capture the markets and application domains targeted by the candidate FPGA architecture. Using an unrepresentative set of benchmarks means optimizing for the wrong targets.

Among the existing open-source benchmark suites, which we will discuss in a later section, none of them focus on (or even capture any) benchmarks from the increasingly important DL domain. Therefore, it becomes very tedious to evaluate architecture and CAD optimizations for DL-targeted FPGAs, since researchers have to first implement their own benchmarks. This limits any research efforts in this direction to only individual isolated ones, and makes it virtually impossible to have meaningful comparisons between different ideas across the FPGA research community. Our work addresses this by presenting Koios[1], an open-source benchmark suite of DL acceleration benchmark circuits for FPGA architecture and CAD research. This suite consists of 19 benchmarks that capture a wide variety of accelerated neural networks, design sizes, numerical precisions, and circuit characteristics. To maximize the utility of these benchmarks, we made them compatible with the Verilog-to-Routing (VTR) flow [8], which is arguably the most widely-used FPGA architecture and CAD research framework. Researchers can use these benchmarks seamlessly with VTR and with minor modifications, can even use them with other toolchains.

Koios benchmarks are representative of modern DL workloads; many of them are re-created from prior works and some are replicas of industrial architectures. In addition to being more pipelined and DSP/BRAM intensive, these benchmarks have higher usage of structures like wide busses, large reduction trees, hard block cascades and large fanouts. This makes Koios benchmarks much better suited for DL-targeted FPGA architecture exploration than any non-DL benchmark suite.

[1]*Koios* (also written as *Coeus*) is the Titan of intelligence in Greek mythology. Unlike the Titan benchmarks, our suite focuses on deep learning.

All the benchmarks along with the FPGA architecture we used for our experiments in this paper are open-sourced as a part of VTR[2]. In this paper, we make the following contributions:

- Introduce the Koios benchmarks and describe the different characteristics of the constituent designs.
- Present the results of running our benchmarks through VTR using an FPGA architecture description file that we develop to capture complex DSP features typical of recent FPGAs.
- Compare circuit statistics to those of the VTR benchmarks to highlight the added value of our new suite.
- Describe two example case studies that use these benchmarks to explore architectural optimizations for DL.

II. Related Work

A. FPGA Benchmark Suites

There are several benchmark suites that were used by FPGA architecture and CAD researchers throughout the past three decades. The classic MCNC20 benchmarks [9] are extremely small and simple designs that do not use any FPGA hard blocks. Therefore, they do not represent modern FPGA use-cases and are rarely used for architecture or CAD studies nowadays. The twenty largest circuits from this suite (often referred to as the Toronto20 [10]) are provided in the input format consumed by the Versatile Place and Route (VPR) tool suite. The UMass RCG HDL Benchmark Collection [11] has larger designs mostly representing DSP applications. However, this suite does not target an open-source FPGA framework. The Groundhog benchmarks [12] are shown to work with academic toolflows and are targeted towards evaluation of power consumption of FPGAs for mobile computing applications. ERCBench [13] is another suite consisting of hybrid hardware/software applications. The designs in this suite represent designs from multimedia, wireless communications and cryptography. They do not contain DL benchmarks, and do not work with academic FPGA tools.

VTR [8] has a suite of benchmarks as well. These VTR benchmarks vary from small (321 netlist primitives) to medium-sized designs (165,809 primitives) and they capture a multitude of applications like image processing, soft processors and arithmetic. The Titan benchmark suite [14] contains modern heterogeneous large designs (90K to 1.8M netlist primitives). However, they target a hybrid CAD flow that is architecture-specific as logic synthesis is performed using the Intel Quartus flow only for the Stratix IV architecture. In contrast to all existing suites, Koios is the only one that provides large, heterogeneous, architecture-agnostic benchmarks that work with a completely open-source flow such as VTR, and focuses on the increasingly important DL domain.

B. DL-Optimized FPGAs

Recently, FPGA vendors have released products with many DL-targeted features to cater to the ever-growing demands of DL workloads. For example, the Xilinx Versal ACAP [15] added specialized vector processors for DL acceleration, and Intel's Stratix 10 NX devices integrated in-fabric AI tensor blocks [3]. In addition, the announced Achronix Speedster7t FPGAs [16] will have embedded machine learning processor (MLP) blocks that tightly couple memory and compute for DL, and the FlexLogix nnMAX [17] inference IP also contains tiles with hardened convolution logic. For their architecture exploration, FPGA vendors typically use proprietary customer designs or internal benchmarks that are not accessible to the research community.

There have also been a number of academic research proposals for optimizing FPGA architectures for DL. Eldafrawy et al. [18] proposed several enhancements to the logic block architecture to pack more arithmetic bits or add a shadow multiplier in them for improved DL performance. They used simple multiplier/MAC and 4×4 matrix multiplication microbenchmarks to evaluate their proposed ideas. In [19], [20], the authors explored enhancing DSP blocks by efficiently supporting low precision multiplications. For these studies, the authors design their own benchmarks to evaluate their ideas. Arora et al. [21] also proposed adding Tensor slices in FPGAs. Again, they use their own designs, a TPU-like overlay and several microbenchmarks, for their evaluation. We believe that an open-source benchmark suite is needed to create a common ground for evaluating and comparing such FPGA architectural enhancements for DL.

III. The Koios Benchmark Suite

Our collection of benchmark designs in the Koios suite come from a multitude of applications within the DL domain. They cover a wide variety of different design sizes, implementation styles, target neural networks, acceleration paradigms, numerical precisions, and circuit properties as summarized by the overview in Table I, and detailed in this section.

- **Design Size**: The smallest design has $11,519$ netlist primitives while the largest has $1,085,877$. Any latch, gate or hard block resulting from logic synthesis counts as a netlist primitive. Some benchmarks, such as `clstm_like`, `dla_like`, `tpu_like`, have multiple size variants (i.e. small, medium, large). In these cases, the size indicates the parallelism factor used in the design. Bigger designs create a more challenging optimization problem for the CAD tools, while smaller ones have faster compilation time suitable for early-stage architecture and CAD experiments.
- **Implementation Style**: Although all the designs in the benchmark suite are provided to users in the form of Verilog HDL implementations, some were originally implemented in RTL while others were automatically generated from higher level language descriptions using high-level synthesis (HLS) tools. HLS-generated designs typically have specific design characteristics that are not generally seen in hand-coded RTL designs, such as widely distributed control signals and complex state machines.
- **Target Neural Network**: Our benchmarks cover all major classes of neural networks. These include: multi-layer per-

[2]https://tinyurl.com/vtrkoios

TABLE I: The Koios Benchmarks (in decreasing order of number of netlist primitives)

Benchmark	Description	Implementation	Network	Precision	Acc. Paradigm	2D Systolic	Winograd/FFT	Reduction	Buffers	DSP usage	Cent. buffers	Based on	Other Properties
clstm_like (S/M/L)	CLSTM-like accelerator	RTL	RNN	int18	Overlay		✓		✓[3]	✓		[22]	Circular compression
dla_like (S/M)	Intel-DLA-like accelerator	RTL	CNN[2]	int8/16	Overlay		✓		✓[3]	✓[4]	✓	[23] [24]	Daisy chain
lstm	LSTM engine	RTL	RNN	int16	Layer			✓	✓	✓			Streaming dataflow
tpu_like (S/M)	Google-TPU-v1-like accelerator	RTL	Any[12]	int8	Overlay	✓			✓	✓	✓	[25]	APB interface
bnn	4-layer binary neural network	HLS	MLP[1]	binary	Custom					✓		[26] [27]	int16 act/norm
tiny_darknet_like	Accelerator for Tiny Darknet	HLS	CNN[12]	fp16	Custom				✓[3]	✓		[28]	Fused layer pairs
gemm_layer	Matrix multiplication engine	RTL	MLP	bfloat16	Layer	✓				✓	✓		AXI interface
attention_layer	Transformer self-attention layer	RTL	RNN	int16	Layer			✓	✓[3]	✓		[29]	GEMV based
conv_layer	GEMM based convolution	RTL	CNN	int16	Layer	✓			✓	✓	✓		3x3 filters
spmv	Sparse matrix vector multiplication	RTL	MLP	int8	Layer				✓	✓	✓	[30] [31]	COO sparsity enc.
robot_rl	Robot+maze application	RTL	RL	int8/16/32	Custom				✓	✓	✓	[32] [33]	Q-learning algo
reduction_layer	Add/max/min reduction tree	RTL	Any	int16	Layer			✓	✓		✓		Reduces 128 inputs
softmax	Softmax classification layer	RTL	Any	fp16	Layer			✓		✓		[34]	LUT based exp/log
conv_layer_hls	Sliding window convolution	HLS	CNN	fp16	Layer				✓	✓			1x1 filters
eltwise_layer	Matrix elementwise add/sub/mult	RTL	Any	bfloat16	Layer				✓	✓	✓		Broadcast heavy

[1] Has Normalization layer [2] Has pooling layer [3] Uses double buffering [4] Has DSP cascade chains

ceptrons (MLPs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), and reinforcement learning (RL). These different classes have different compute and memory requirements, which reflects on the resource breakdown and routing patterns of their corresponding benchmark circuits. Some designs are also generic and can be used to accelerate any type of network.

- **Acceleration Paradigm**: FPGAs are used for acceleration of DL workloads in different ways. One way is to design a flexible software-programmable overlay architecture that can execute different DL models without the need to reprogram the FPGA with a new bitstream similar to the Microsoft Brainwave [35] architecture. These designs tend to have instruction decoders and more complicated control logic to enable this level of flexibility. In other cases, a custom network-specific architecture is mapped to an FPGA to maximize efficiency similar to the approach used in [1]. The control logic of these circuits is usually hard-coded and implemented as relatively simple state machines. Another approach is to implement layer-specific accelerators that are invoked by software running on the host CPU. These circuits are mostly streaming-style datapaths with simple or even no control paths. Our benchmark suite contains designs from all three acceleration paradigms.
- **Numerical Precisions**: One of the main advantages of using FPGAs to accelerate DL workloads is the ability to design hardware for custom numerical precisions, which is a commonly used technique in accelerating DL workloads [36]. The designs in our suite use various precisions, including: binary (`bin`), different fixed point types `int8/16/32`, brain floating point (`bfloat16`) [37], and IEEE half-precision floating point (`fp16`). The diversity in the benchmarks' numerical precisions is useful for exploring new reconfigurable DSP block architectures and different hard arithmetic circuitry.
- **Circuit Properties**: Our benchmarks have varying circuit styles that can potentially exercise different components of the CAD tools in different ways. For example, regular structures like systolic arrays can be used for optimizing placement algorithms, large reduction trees can form local routing congestions that stress the routing algorithms, long cascades of hard blocks impose harder placement constraints, etc. The benchmarks are also highly heterogeneous (i.e. use different types of FPGA resources) with varying degrees as will be discussed in Section V.

These benchmarks are implemented and compiled together in this suite with the intention to be used for FPGA architecture exploration and CAD tool optimization. They aim to accurately capture all these different circuit structures and compositions, but should not be expected to be deployed as standalone functional systems. We are confident that these circuits are structurally correct and tried to verify their high-level functionality to the best of our ability. However, full functional verification on many different test cases is out of the scope of this work.

IV. Methodology

A. Ensuring VTR Compatibility

The designs in the benchmark suite are implemented and tested first using commercial FPGA tools from Xilinx and Intel for ease of development and debugging. Then, we performed several modifications to these designs to ensure their compatibility with the VTR flow. VTR uses Odin II, an academic open source synthesis tool, as its conventional front-end. To work around the Verilog support limitations of Odin II and at the same time maintain the conventional fully open-source VTR flow, we implemented several scripts to help automate the process of replacing unsupported Verilog constructs (e.g. `signed`, `integer` variables, `generate` for loops, unpacked arrays, etc.) with alternative/unrolled Verilog constructs that are supported by Odin II. In addition, vendor-specific and architecture-specific IP cores (e.g. floating

point adders and multipliers, RAM macros) were replaced with ones that are compatible with VTR and the FPGA architecture file used for our experiments. This process was especially challenging for the designs generated from HLS tools which tend to be non-human-readable in many cases. Several improvements to the language coverage and reported error messages of Odin II are continuously being implemented to mitigate such challenges for future research efforts.

B. Experimental Setup

We use the most-recent VTR 8.0 version [8] for all our experiments in this paper. While running VTR, we provide an SDC (Synopsys Design Constraints) file in which the target clock frequency is set to 0 (i.e. VTR will optimize the design for maximum clock frequency). We also disable timing analysis for paths to/from the FPGA IOs. For all experiments, we run VTR with auto layout enabled (meaning the grid size expands based on the resources required by the design), the default timing-driven routing option with a maximum of 150 routing iterations, and a fixed channel width of 300 wires. All reported results are the average of runs using 3 different seeds. For experiments in which we report VTR flow runtime and peak memory usage, we use an Intel Xeon CPU E5-2430 running at 2.5 GHz with 64 GB of memory.

One of the main motivations of this work is to compare various properties of our Koios benchmarks with other existing non-DL-targeted benchmarks that are commonly used to drive FPGA architecture and CAD research. The most relevant suite for comparison is the VTR benchmark suite, because these are compatible with the same fully open source VTR flow. Other existing suites are either too small and do not represent realistic modern use cases of FPGAs or depend partially on commercial CAD tools. For these comparative experiments, we only use the VTR benchmarks with more than $10,000$ netlist primitives, which is a common practice in CAD-related studies [38]. Designs smaller than that are not representative of realistic benchmarks and they cannot be used to derive any reliable conclusions.

C. FPGA Architecture Description

We develop a new FPGA architecture description file to capture some relevant features of modern FPGAs. This architecture description file will be open sourced along with the benchmark suite. The delays and areas of all the FPGA blocks, including the DSP tiles, are obtained from COFFE [39] using a 22nm technology node from PTM [40]. The circuits in this architecture are optimized for area-delay product which leads to relatively higher delays compared to performance-optimized commercial FPGAs such as the Arria 10 family. The rest of this subsection describes the details of the FPGA architecture that we develop and use for all our experiments.

1) Floorplan: The FPGA contains columns of logic blocks, DSPs and block RAMs (BRAMs). Both DSP and BRAM columns repeat every 16 columns and are interleaved such that every 8th column is a DSP or a BRAM. The DSP and BRAM tiles are 4 and 2 rows high, respectively. IO pads are arranged along the perimeter of the FPGA.

2) Routing Architecture: The architecture uses unidirectional routing with wire segments of length 4 (260 out of 300 wires) and length 16 (40 out of 300 wires). The length 16 wires do not directly connect to block pins and are only accessible from the length 4 wires. Switches appear after every 4 blocks on the length 16 wires. The switch blocks use a custom switching pattern based on the Stratix-IV-like architecture used in the Titan flow [14]. The input and output flexibility of connection blocks are set to 0.15 and 0.1, respectively.

3) Logic Blocks: Each logic block (LB) contains 10 basic logic elements (BLEs) similar to that in the Intel Stratix-10-like architecture from [18]. Each block has 60 input pins, 40 output pins, and a 50% sparsely populated local input crossbar. Each BLE has a 6-input LUT which can be fractured into two 5-input LUTs. The BLE also has 2 flip-flops and 2 bits of arithmetic with dedicated carry chains between LBs. Each BLE has 8 inputs and 4 optionally registered outputs.

4) DSP Slices: This architecture has a complex DSP block that supports most of the operating modes in the state-of-the-art Intel Agilex DSP block [41]. Multiple fixed point (9x9, 18x19, 27x27) and floating point (IEEE 32-bit (`fp32`), IEEE 16-bit (`fp16`) and Brain floating point (`bfloat16`)) precisions are supported. In addition, the DSP block has dedicated output chains for cascading several DSP blocks in the same column for efficient dot product structures.

5) BRAMs: BRAM blocks have a capacity of 20 Kilobits and have registered inputs and outputs. True and simple dual port modes are supported. In the simple dual port mode, a BRAM can be configured as: 512×40, 1024×20 and 2048×10, while in true dual port mode it can be configured only as: 1024×20 and 2048×10. The delays and areas of a BRAM block are obtained by interpolation between the values obtained from COFFE for a 16 Kilobit BRAM and a 32 Kilobit BRAM.

Some benchmarks in Koios use advanced DSP features that are available in this FPGA architecture by instantiating DSP macros to implement native `fp16` multiplications or use the hard dedicated chains. These modes are architecture-specific; however, users can simply replace the macro instantiations in our benchmarks with their equivalents for different architectures. In addition, we also include alternative versions of the benchmarks (using `` `ifdef..`endif ``) implementing the same functionality with behavioral Verilog that is automatically mapped to the FPGA soft logic when an architecture without the required macro definitions is used.

Koios benchmarks can be used to explore FPGA architectural modifications involving adding new hard blocks to FPGAs, similar to some recent DL-optimized FPGAs [3] [21]. This can be done by: (1) modifying the synthesis engine to extract specific patterns from the Verilog design and map them to the new blocks, or (2) modifying the benchmarks to instantiate these new blocks (defined in the VTR architecture file).

TABLE II: VTR results of the Koios benchmarks.

Benchmark	Netlist Primitives	Logic Depth	Used IOs	Used LBs	Used DSPs	Used BRAMs	Max. Freq.	Routed Wirelength	Elapsed Time	Peak Memory
clstm_like (L)	1,085,877	3	1,159	25,995	**962**	1,161	110.2	5,534,505	1,171.4	12,658.4
clstm_like (M)	745,829	3	871	17,641	**662**	784	115.4	3,612,133	560.6	8,691.1
dla_like (M)	609,180	5	411	11,359	400	**1,008**	125.9	3,349,783	260.7	6,009.7
clstm_like (S)	405,776	3	583	9,309	**362**	407	127.6	1,744,947	152.8	4,679.1
dla_like (S)	269,040	5	207	5,545	128	**828**	147.7	1,475,558	86.1	4,304.7
lstm	249,841	7	36	6,626	**610**	305	121.6	1,828,974	308.2	5,892.8
tpu_like (M)	244,884	5	1,188	4,255	**1,064**	26	98.62	2,412,297	156.1	9,163.1
bnn	204,601	3	382	**5,695**	63	0	126.8	1,233,543	20.9	2,153.1
tiny_darknet_like	154,096	6	46	7,417	106	**3,978**	63.9	3,033,846	571.1	16,253.5
tpu_like (S)	67,086	5	644	1,134	**276**	14	124.8	579,437	31.9	2,507.5
gemm_layer	64,792	4	1,779	1,989	**200**	0	308.1	717,412	25.4	1,982.2
attention_layer	45,342	7	1,074	1,248	**105**	161	132.2	370,030	16.7	1,152.3
conv_layer	45,039	4	156	1,185	**84**	56	166.1	293,011	9.4	876.3
spmv	28,505	6	19	885	32	**257**	167.9	275,500	14.5	1,492.8
robot_rl	28,080	15	387	1,324	18	**96**	83.6	228,378	9.1	549.5
reduction_layer	18,323	6	54	805	0	**52**	141.7	183,739	2.2	363.2
softmax	13,189	10	552	518	**53**	0	112.2	127,704	2.5	513.3
conv_layer_hls	12,093	3	**3,299**	1,715	12	21	164.7	112,362	19.2	8,929.1
eltwise_layer	11,519	4	249	348	**48**	72	174.9	170,857	2.1	480.9

Frequency is in MHz, Routed Wirelength is in units of length 1 segments, Elapsed Time is in minutes, and Peak Memory is in MBs.

V. Benchmark Results

A. Properties of Koios benchmarks

Table II shows the main VTR results for the Koios benchmarks when running them with the FPGA architecture described in Section IV-C.

The results show that these designs, with sizes ranging from 11K to 1M netlist primitives, are deeply-pipelined with 12 out of the 19 benchmarks having critical paths with 5 or less logic levels on them. The benchmarks are also highly diverse in heterogeneity, with varying circuit compositions between soft logic, DSPs, and BRAMs. For example, some designs do not utilize any BRAMs since they either implement only the workload datapath (e.g. `gemm_layer` and `softmax`) or use distributed registers for storage (e.g. `bnn`). On the other hand, there are other BRAM-intensive designs such as `tiny_darknet_like` with close to $4,000$ BRAMs utilized. Similarly with DSPs, there are some designs that use very few or no DSPs (e.g. `bnn` and `reduction_layer`) as they mostly implement other non-multiplication operations in DL workloads such as pop-count or max/min/add reduction. Other designs are DSP-intensive (e.g. large `clstm_like` and medium `tpu_like`) with around $1,000$ DSP blocks. Table II also shows that different types of resources are the grid-size limiting factor for different benchmarks in our suite. The majority of the designs are bound by hard blocks, as indicated by the bold entries in the table, which emphasizes that these benchmarks can be useful for exploring new DSP and BRAM architectures.

Most of the designs in the Koios suite can achieve reasonably high operating frequencies up to 308 MHz and an average of 137 MHz. The FPGA architecture used for our experiments is not very fast. The delays in the architecture are based on area-delay-optimized PTM models (with raw delays similar to 40 nm Stratix-IV). Changing the delays of FPGA resources to those typical of a high-speed ($\leq$14 nm) device would increase the frequency by $>2\times$. The `tiny_darknet_like` design is a clear outlier with a frequency of 63.9 MHz since the grid size required to implement this circuit was significantly expanded due to the large number of BRAMs needed. This resulted in some very long paths between BRAMs and soft logic flip-flops (FFs). The total routed wirelength of the benchmarks are largely correlated with the circuit size and ranges from 171K up to 5.5M units of length 1 wire segments. Fig. 1 plots the VTR flow runtime for each of the Koios benchmarks as listed in Table II. It shows that the runtime grows quadratically with the number of netlist primitives in the circuits.

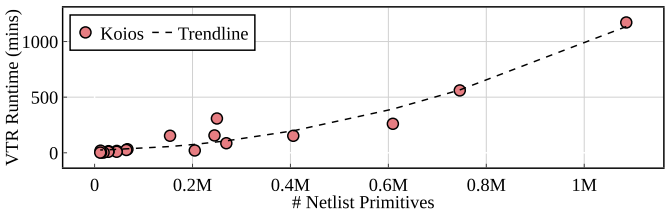

Fig. 1: VTR runtime for the Koios benchmarks.

B. Comparison to the VTR Benchmarks

Fig. 2a shows a scatter plot of the DSP and BRAM to LB ratios for both Koios (red) and VTR (blue) benchmarks as metrics for their DSP and memory density. The individual ratios for each of the benchmarks are shown by ($\times$) symbols while the average across the whole benchmark suite is marked by the stars. The figure shows that, on average, the Koios benchmarks are more DSP and memory rich than the VTR benchmarks. The Koios suite has a $1.8\times$ and $4.7\times$ higher DSP to LB and BRAM to LB ratios, respectively. The individual benchmarks of the Koios suite are also more scattered and varying across the spectrum of DSP and BRAM compositions. More importantly, it shows that most of the VTR benchmarks have very low DSP and BRAM densities (except for the only `stereovision2` outlier circuit), making them inadequate for evaluating any DSP or BRAM architecture modifications.

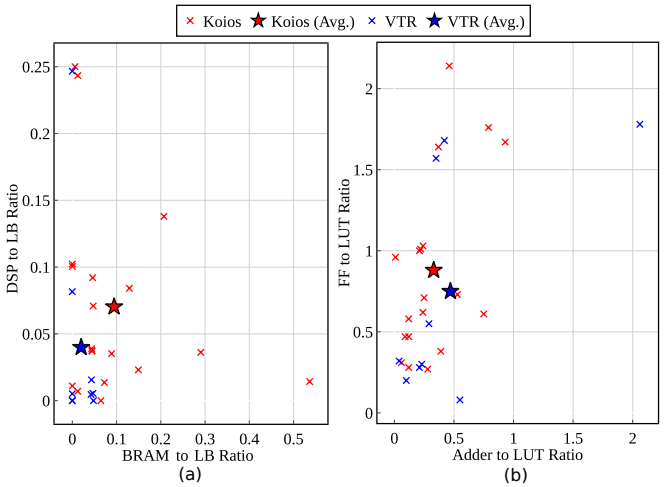

Fig. 2: Comparing circuit compositions of Koios & VTR benchmarks: (a) DSP/BRAM to LB ratios, (b) FF/adder to LUT ratios.

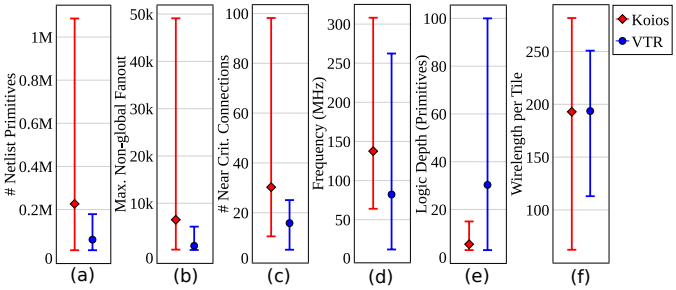

Fig. 3: Averages and ranges of key metrics of Koios & VTR suites.

Fig. 2b has a similar plot for FF and single-bit adder to LUT ratios. It shows that the Koios suite has 1.17× higher ratio between FFs and LUTs which reflects their deeply-pipelined nature, and 30% lower adder to LUT ratio compared to the VTR suite. However, the average adder to LUT ratio of the VTR suite is significantly skewed by a single benchmark (`stereovision2`) which has 60,753 1-bit adders and only 29,541 LUTs. If we exclude this outlier, the Koios suite has a 1.2× higher average adder to LUT ratio.

Fig. 3 illustrates averages and ranges of key metrics for both Koios and VTR benchmark suites. Fig. 3a-d show that the Koios benchmarks have 3.7× more netlist primitives, 6.5× larger non-global fanouts, 1.9× more near (top 10%) critical connections, and 1.7× higher frequencies on average compared to the VTR benchmarks. The Koios benchmarks are also scattered across a much wider range of values for each of those metrics. Fig. 3e shows that the Koios designs have an average of 5 logic levels on the critical path, compared to 30 levels for the VTR benchmarks. This also reflects the deeply-pipelined nature of our benchmarks which is a key property of modern FPGA designs. Fig. 3f shows that the two benchmark suites have similar average routed wirelength per tile, with the most wiring dense circuit in Koios having 12% higher wirelength per tile compared to the most-wiring dense circuit in the VTR suite.

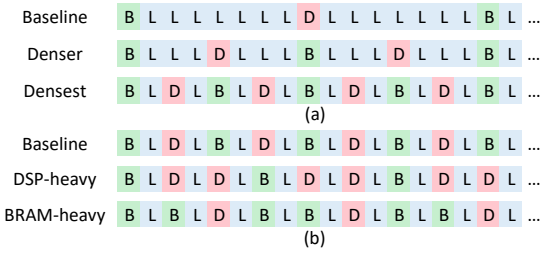

Fig. 4: FPGA layouts the architectures used in our case studies.

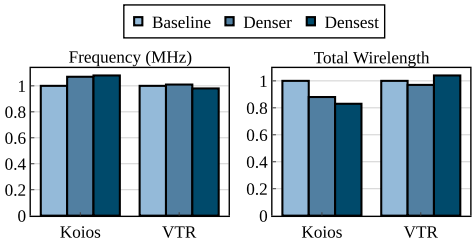

Fig. 5: Effect of varying the density of DSPs and BRAMs on Koios and VTR benchmark suites.

VI. Architecture Exploration Case Studies

Our Koios benchmark suite is architecture-agnostic and does not depend on any commercial tools for any portion of the FPGA CAD flow. Thus, it enables the use of these benchmarks to perform flexible FPGA architecture exploration using the fully-open-source VTR flow. In this section, we perform two example case studies to demonstrate that.

A. Case Study 1: Hard Blocks to Soft Logic Ratio

As shown in Table II, our DL-focused circuits are highly heterogeneous (i.e. DSP and BRAM intensive). Thus, in our first case study, we vary the density of these hard blocks with respect to soft logic. We experiment with 3 different density levels, as shown in Fig. 4a, with 1:7, 1:3, and 1:1 ratio between hard block and soft logic columns for the baseline, denser, and densest architecture variations, respectively. We evaluate all three architecture variations using both the Koios and VTR benchmarks. Fig. 5 shows the geomean frequency and total routed wirelength for both suites. For the DL-oriented Koios benchmarks, the frequency increases and wirelength decreases as the density of hard blocks increases. Since these benchmarks heavily utilize these blocks, increasing their density in the FPGA grid brings them closer to each other, which in turn reduces the critical paths and total length of used wires. The densest architecture variation results in 8% increase in frequency and 17% reduction in total wirelength on average across all benchmarks in the Koios suite. For the VTR benchmarks, both frequency and wirelength are slightly improved for the denser variation (1% higher frequency and 3% lower wirelength), before getting worse for the densest architecture. These results show that a higher density of DSPs and BRAMs is favorable for building DL-optimized FPGAs, at the cost of a slight or no degradation in QoR for the general VTR benchmarks (in the densest and denser architecture variations respectively).

TABLE III: Effect of varying the FPGA's DSP to BRAM ratio.

Metric	Arch.	Geo-mean	DSP-heavy tpu_like(M)	BRAM-heavy tiny_darknet_like
Freq.	**Baseline**	141.2	141.1	94.1
	DSP-heavy	141.9	**153.3**	86.8
	BRAM-heavy	140.8	120.4	**101.1**
WL	**Baseline**	622,189	1,460,366	2,076,993
	DSP-heavy	623,777	**1,325,930**	2,313,599
	BRAM-heavy	641,263	1,661,778	**1,944,531**
Grid	**Baseline**	84×84	134×134	180×180
	DSP-heavy	85×85	**116×116**	220×220
	BRAM-heavy	88×88	164×164	**156×156**

Frequency is in MHz, Wirelength (WL) is in units of length 1 wires.

B. Case Study 2: DSP to BRAM Ratio

In our first case study, we varied the ratio of hard blocks to soft logic while keeping a fixed 1:1 DSP to BRAM ratio. For the second case study, we carry over the best architecture variation for DL benchmarks from the first case study (i.e. densest). However, we vary the DSP to BRAM ratio between 2:1 and 1:2 to create DSP-heavy and BRAM-heavy variations respectively (in addition to the baseline with 1:1 ratio), as shown in Fig. 4b. Table III presents the results of this experiment. It shows the geomean frequency, routed wirelength, and FPGA grid size for the whole Koios suite, as well as the results for a DSP-intensive benchmark (medium `tpu_like`) and a BRAM-intensive benchmark (`tiny_darknet_like`). The geomean results do not show a strong trend that clearly favors a specific architecture. However, we observe that the DSP-heavy `tpu_like` design has 8.6% higher frequency, 9.3% lower wirelength, and requires a 25% smaller chip when implemented on the DSP-heavy architecture compared to the baseline. It also performs considerably worse on all metrics when implemented on a BRAM-heavy architecture. Similarly the BRAM-heavy `tiny_darknet_like` benchmark has 7.5% higher frequency, 6.4% lower wirelength, and requires a 25% smaller chip when implemented on the BRAM-heavy architecture compared to the baseline. These experiments highlight that Koios strikes a good balance between different circuit compositions and can be reliably used for DL-optimized FPGA architecture exploration.

VII. Conclusion

In this paper, we presented Koios, a DL-focused benchmark suite for FPGA architecture and CAD research. This suite is a diverse collection of 19 curated benchmarks covering various facets of the DL acceleration landscape. We first introduce the different benchmarks in the suite and highlight their diversity. We then present results of running these benchmarks through the VTR flow and compare them to the existing non-DL VTR benchmarks. Finally, we present two example case studies for DL-optimized FPGA architecture exploration using these benchmarks. The Koios suite is open-sourced as a part of VTR and we highly encourage the FPGA community to contribute to this benchmark suite to help build a better and bigger set of DL benchmarks that can guide the design of future FPGA architectures and CAD algorithms.

Acknowledgement

We would like to thank Helen Dai and Zach Zheng from the University of Toronto for contributing to the benchmarks. We are grateful to the National Science Foundation (grant number 1763848), the NSERC/Intel Industrial Research Chair in Programmable Silicon, the Vector Institute for AI, and the Intel/VMWare Crossroads Research Center for funding support. Any opinions, findings, conclusions or recommendations are those of the authors and not of the funding institutions.

References

[1] M. Hall and V. Betz, "HPIPE: Heterogeneous Layer-Pipelined and Sparse-Aware CNN Inference for FPGAs," *arXiv preprint arXiv:2007.10451*, 2020.

[2] A. Boutros *et al.*, "Beyond Peak Performance: Comparing the Real Performance of AI-Optimized FPGAs and GPUs," in *International Conference on Field Programmable Technology (FPT)*, 2020.

[3] M. Langhammer *et al.*, "Stratix 10 NX Architecture and Applications," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021.

[4] E. Nurvitadhi *et al.*, "Why Compete When You Can Work Together: FPGA-ASIC Integration for Persistent RNNs," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019.

[5] S. Ahmad *et al.*, "Xilinx First 7nm Device: Versal AI Core (VC1902)," in *Hot Chips Symposium*, 2019.

[6] J. Zhang *et al.*, "Frequency Improvement of Systolic Array-Based CNNs on FPGAs," in *International Symposium on Circuits and Systems (ISCAS)*, 2019.

[7] A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression," *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 4–29, 2021.

[8] K. E. Murray *et al.*, "VTR 8: High Performance CAD and Customizable FPGA Architecture Modelling," *ACM Transactions on Reconfigurable Technology Systems (TRETS)*, vol. 13, no. 2, 2020.

[9] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," 1991.

[10] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," in *International Conference on Field-Programmable Logic and Applications (FPL)*, 1997.

[11] J. Allen. (2006) UMass RCG HDL Benchmark Collection. [Online]. Available: http://www.ecs.umass.edu/ece/tessier/rcg/benchmarks/

[12] P. Jamieson *et al.*, "Benchmarking and Evaluating Reconfigurable Architectures Targeting the Mobile Domain," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 15, no. 2, 2010.

[13] D. Chang *et al.*, "ERCBench: An Open-Source Benchmark Suite for Embedded and Reconfigurable Computing," *International Conference on Field Programmable Logic and Applications (FPL)*, 2010.

[14] K. E. Murray *et al.*, "Timing-Driven Titan: Enabling Large Benchmarks and Exploring the Gap between Academic and Commercial CAD," *ACM Transactions on Reconfigurable Technology Systems (TRETS)*, vol. 8, no. 2, 2015.

[15] Xilinx, Inc. (2018) Xilinx AI Engines and Their Applications. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp506-ai-engine.pdf

[16] Achronix Semiconductor. (2019) Speedster7t FPGAs. [Online]. Available: https://www.achronix.com/product/speedster7t/

[17] Flex Logix Technologies, Inc. (2019) Flex-Logix nnMAX Inference Acceleration Architecture. [Online]. Available: https://flex-logix.com/wp-content/uploads/2019/09/2019-09-nnMAX-4-page-Overview.pdf

[18] M. Eldafrawy *et al.*, "FPGA Logic Block Architectures for Efficient Deep Learning Inference," *ACM Transactions on Reconfigurable Technology Systems (TRETS)*, vol. 13, no. 3, 2020.

[19] A. Boutros *et al.*, "Embracing Diversity: Enhanced DSP Blocks for Low-Precision Deep Learning on FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2018.

[20] S. Rasoulinezhad *et al.*, "PIR-DSP: An FPGA DSP Block Architecture for Multi-precision Deep Neural Networks," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019.

[21] A. Arora *et al.*, "Tensor Slices to the Rescue: Supercharging ML Acceleration on FPGAs," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021.

[22] S. Wang *et al.*, "C-LSTM: Enabling Efficient LSTM Using Structured Compression Techniques on FPGAs," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2018.

[23] U. Aydonat *et al.*, "An OpenCL Deep Learning Accelerator on Arria 10," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017.

[24] A. Boutros *et al.*, "You Cannot Improve What You Do Not Measure: FPGA vs. ASIC Efficiency Gaps for Convolutional Neural Network Inference," *ACM Transactions on Reconfigurable Technology Systems (TRETS)*, vol. 11, no. 3, 2018.

[25] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," 2017.

[26] J. Duarte *et al.*, "Fast Inference of Deep Neural Networks in FPGAs for Particle Physics," *Journal of Instrumentation*, vol. 13, no. 07, 2018.

[27] J. Ngadiuba *et al.*, "Compressing Deep Neural Networks on FPGAs to Binary and Ternary Precision with hls4ml," *Machine Learning: Science and Technology*, vol. 2, no. 1, 2020.

[28] J. Redmon. (2018) Tiny darknet. [Online]. Available: https://pjreddie.com/darknet/tiny-darknet/

[29] A. Vaswani *et al.*, "Attention is All You Need," in *International Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[30] J. Fowers *et al.*, "A High Memory Bandwidth FPGA Accelerator for Sparse Matrix-Vector Multiplication," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2014.

[31] Xilinx. (2017) Gemx. [Online]. Available: https://github.com/Xilinx/gemx

[32] S. Spanò *et al.*, "An Efficient Hardware Implementation of Reinforcement Learning: The Q-Learning Algorithm," *IEEE Access*, vol. 7, 2019.

[33] L. Da Silva *et al.*, "Parallel Implementation of Reinforcement Learning Q-Learning Technique for FPGA," *IEEE Access*, vol. 7, 2019.

[34] Z. Wei *et al.*, "Design Space Exploration for Softmax Implementations," in *International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2020.

[35] J. Fowers *et al.*, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in *International Symposium on Computer Architecture (ISCA)*, 2018.

[36] B. Darvish Rouhani *et al.*, "Pushing the Limits of Narrow Precision Inferencing at Cloud Scale with Microsoft Floating Point," *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[37] S. Wang and P. Kanwar. BFloat16: The Secret to High Performance on Cloud TPUs. https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus.

[38] M. Elgammal *et al.*, "Learn to Place: FPGA Placement Using Reinforcement Learning and Directed Moves," in *International Conference on Field Programmable Technology (FPT)*, 2020.

[39] S. Yazdanshenas and V. Betz, "COFFE2: Automatic Modelling and Optimization of Complex and Heterogeneous FPGA Architectures," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, 2019.

[40] Arizona State University. (2012) Predictive Technology Model. [Online]. Available: http://ptm.asu.edu/

[41] Intel. (2019) Intel Agilex FPGAs and SOCs. [Online]. Available: https://www.intel.com/content/www/us/en/products/programmable/fpga/agilex.html