Scaling the Cascades: Interconnect-aware FPGA implementation of Machine Learning problems

Ananda Samajdar[2], Tushar Garg[1], Tushar Krishna[2], Nachiket Kapre[1]

[1]School of Electrical and Computer Engineering
University of Waterloo
Ontario, Canada
t3garg,nachiket@uwaterloo.ca

[2]School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, USA,
anandsamajdar,tkrishna3@gatech.edu

***Abstract*—DSP48s, BRAMs and URAMs in the Xilinx Ultrascale+ family support dedicated cascade interconnect for high frequency, nearest-neighbor data movement using hard wiring resources. We demonstrate how to leverage these interconnect structures to effectively support data movement requirements of dense machine learning (ML) workloads at URAM-limited 650 MHz frequency (714 MHz reported by Vivado). We reformulate convolution and matrix-vector multiplication operations to make effective use of cascade interconnect (1) in DSP48s for supporting the common multiply-accumulate chains, and (2) in BRAMs, and URAMs to exploit the data movement and reuse patterns of ML workloads. The use of these dedicated cascade interconnect are an alternative to Versal AI cores that throw away FPGA flexibility in favor of rigid ASIC components with unproven long-term value. Our 650 MHz operation on the Xilinx VU37P UltraScale+ FPGA is competitive with the 720 MHz state-of-the-art Xilinx SuperTile design. We use 100% URAM288s, 95% DSP48s, and 77% BRAM in contrast to the 100% URAM288s, 56% DSP48, and 40% BRAM usage of the Xilinx SuperTile array. As a result, we deliver a $\approx 7\times$ superior GoogLeNet inference latency while sacrificing 25% of inference throughput than their design. For MLPerf benchmarks we note inference latencies between 3μs–1.89 ms with corresponding throughputs between 528–260K inf/s.**

I. Introduction

Machine learning (ML) acceleration is of great interest to the FPGA community today. Modern FPGAs support massive fine-grained parallelism thanks to programmability, customizability, and control over hardware organization needed to process these workloads. The flexibility and abundance of FPGAs in the Microsoft Azure datacenter has led to accelerator architectures such as the Brainwave [6] that distribute the ML workloads across hundreds of FPGAs. Xilinx has developed a highly-optimized 720 MHz SuperTile [29], [30] systolic computing overlay for acceleration of neural networks using off-the-shelf Xilinx UltraScale+[15], [28] FPGAs, as well as a new series of Versal [32] FPGA architectures that support novel hardened features such as dedicated AI engines and custom interconnect and data movement blocks. ML computations lend themselves well to parallelization but are characterized by massive compute demands, high memory storage and bandwidth requirements, and a quixotic mix of communication requirements.

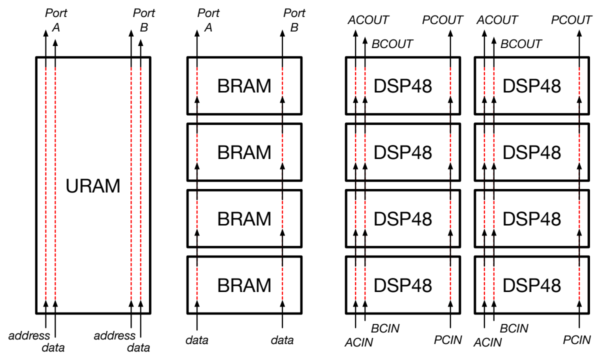
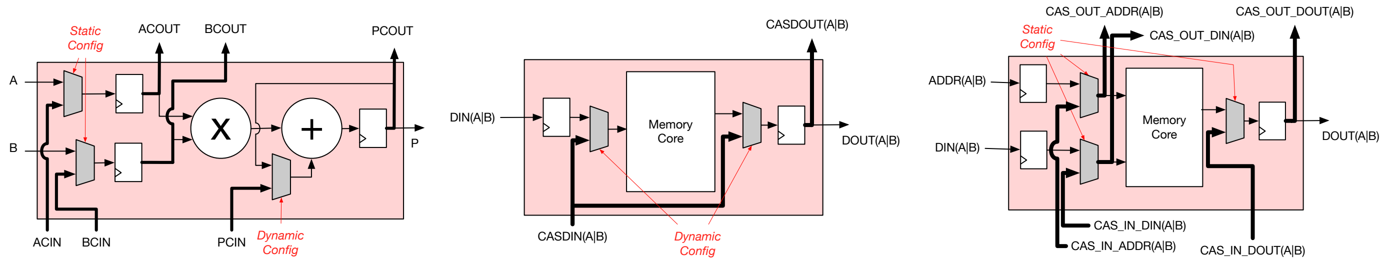


Fig. 1: High-level view of of cascade connections between DSP48, RAMB18, and URAM288 blocks.

The key idea explored in this paper is the explicit use of dedicated interconnect structures available on Xilinx UltraScale+ FPGAs to deliver 650 MHz operation while utilizing 100% URAMs, 95% DSPs, and 77% BRAMs. Contemporary FPGA ML acceleration research focuses on arithmetic enhancements [4] while we focus on high-frequency data movement optimizations. In Figure 1, we show a cartoon representation of the cascade structures used in this paper. The DSP48 arithmetic blocks provide the well-known adder chain cascade structure [7] to accumulate sums, while also providing an opportunity to shift input operands using dedicated wiring as well. The BRAM and URAM memory storage elements can be chained together to create larger communication networks [8] as well as, deeper, more flexible memory structures. Apart from providing the cascade wiring, Xilinx also makes them *programmable*: we can configure their use either statically at compile-time, or in some cases dynamically at runtime. With the programmability features, the wiring can be purposed to support the particular data movement and reuse pattern available in the ML workloads. This mapping strategy allows offloading bulk of the communication demands away from the soft bit-level programmable interconnect onto hard wiring resources. It also makes it possible to develop high-speed FPGA layouts that keep soft interconnect utilization low, and helps the design operate at component maximum frequency.



(a) DSP48 cascade (891 MHz) (b) RAMB18 cascade (825 MHz)) (c) URAM288 cascade (650 MHz)

Fig. 2: Hard Cascade structures embedded in Xilinx hard blocks.

The key contributions of this work are:

1. We develop Xilinx UltraScale+ FPGA overlay accelerators for ML workloads targeting convolution and matrix-vector multiplication as the key building blocks.
2. We show how to directly exploit the dedicated interconnect features of the DSP48, BRAM, and URAM components to map unique data reuse and data movement requirements of ML workloads.
3. We evaluate the efficiency and performance of our architecture across the standard MLPerf [1] benchmark set.
4. We deliver URAM-limited 650 MHz operation on chip-spanning designs mapped to the Xilinx VU37P FPGA using a formulaic floorplanning of the hard blocks like URAMs, DSPs, and BRAMs and resulting short interconnect lengths.

II. Background

In this section we will discuss two key computation libraries in modern ML workloads and review available features of the cascade interconnect on the Xilinx UltraScale+ family.

A. Compute Libraries in ML Workloads

Convolutions and Matrix-Vector multiplications are two key basic linear algebra (BLAS) routines used pervasively in modern ML workloads. A convolution operation slides a "kernel" of weights over an image. Each output is a dot product of the kernel and the input pixels covered by the kernel. 2D convolutions are used extensively in deep networks performing image classification. A matrix-vector multiplication is the key operation in RNNs and MLPs. Many CNN accelerators directly implement convolutions to exploit input/filter reuse during the sliding window operation, unlike GPUs [26] that lower convolutions to matrix-matrix multiplications through extensive replication of data.

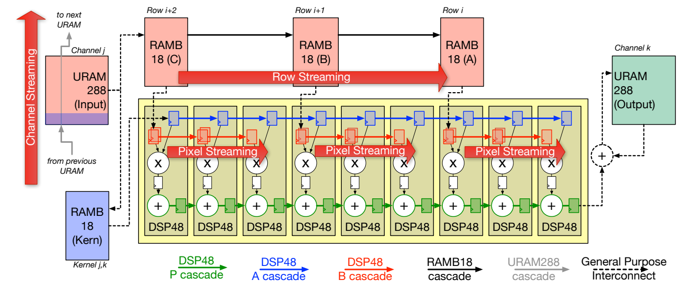
B. Data Reuse within DNN Accelerators

For the uninitiated, ML accelerators implemented either as ASICs or FPGA overlays comprise of a collection of processing elements (PE) to extract parallelism. Each PE can be thought of a small computing engine with some near memory. Variation in mapping strategies [5] of the ML workloads over the accelerator results in data reuse and significantly distinct data access patterns, which in turn affect, performance, power and memory footprint. Such affects are particularly accentuated in the case of deep neural networks. A careful mapping of computing that takes into account the underlying reuse patterns can radically improve the performance on a given device. For our work, we adopt the "weight stationary" approach for exploiting row-streaming reuse during convolution phases, and the "input stationary" strategy for multicasting the vectors during matrix-vector multiplication phases.

C. Dedicated Cascade Interconnect

The Xilinx UltraScale+ device family [15], [28] integrates thousands of hard resources such as DSP and RAM blocks in a columnar arrangement. We enumerate the salient features of the different blocks below:

- **DSP48 (Figure 2a)**: These components primarily support arithmetic integer operations including 27×18 multiplication, and 48b accumulation. A unique feature of the Xilinx DSPs is that they expose configurability within the DSP block to the FPGA logic fabric for runtime control. A developer may not only choose the kind of operation being performed in the DSP block, but also change data routing and data movement pathways within the DSP. The key feature of the DSP48 blocks we wish to exploit in this paper is the ability of multiple DSP blocks to cascade together in a chain-like configuration. This is supported either for performing accumulation of a series of partial products (adder cascade), or for permitting efficient data reuse for certain inputs (input cascade). While the adder cascade is programmable dynamically, the input cascade is only statically configured.
- **RAMB18 (Figure 2b)**: Modern FPGAs provide access to thousands of small distributed on-chip memories that have configurable port widths, and other statically configurable operating modes. A particularly unique feature of the Xilinx UltraScale FPGAs is the presence of nearest-neighbor, dynamically cascadable connections for the two data ports in the same direction as the DSP cascades (uphill). This allows the developer to construct deeper memory structures, cascaded FIFOs, and other user-configurable dataflow patterns. The multiplexers controlling dataflow are only available on the data ports.
- **URAM288 (Figure 2c)**: UltraScale+ FPGAs introduced higher density SRAM blocks with 288kb capacity that sacrifice port width flexibility for lower cost. While the port aspect ratios are not programmable, there is still a column-spanning cascade network for the data ports along


(a) Design of a 3×3 convolution block. DSPs configured in SIMD=2 mode, a set of 8b weights are shifted into the B cascade. One stream of 2×8b = 16b data streamed into the A cascade from different rows. BRAM cascades also configured to exploit row reuse.

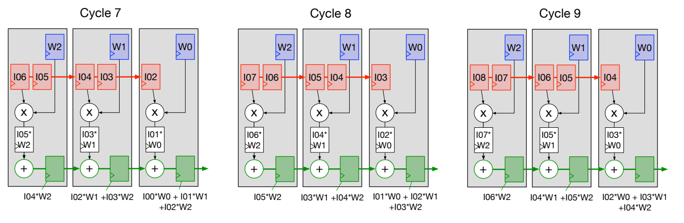

(b) Systolic DSP48 dataflow for 1×3 filter slice using 3 DSP48s.

Fig. 3: Implementing 3×3 Convolution built using a combination of DSPs, BRAMs, and URAMs and hard cascade interconnect in systolic mode.

with address. This allows the developer to address any location in any URAM block in a column with ease. In contrast to BRAM cascades, the URAM cascade network separates the read and write ports into independent cascades. Importantly, it provides cascadeability for data, address, as well as control signals.

III. High-Frequency FPGA Cascades

The architecture of FPGA-based ML accelerator in this paper exploits (1) the resource balance constraints of the device, and (2) unique cascade interconnect features of the UltraScale+ family. In this section, we first discuss the building **blocks** for Convolution and Matrix-Vector multiplication blocks and show how to map these over the cascades to create repeating **tiles** that balance capacity, bandwidth, and precision. After that, we provide an overview of the design space of possible implementations. Pooling operations are mapped onto the same resources as the Convolution engines, while ReLU softmax operations are provided as bypassable operators prior to data commit.

A. Building Block: Convolution

We present a weight-stationary implementation of convolution that takes advantage of data reuse patterns in convolution without transforming it to memory-hungry matrix-matrix operations as discussed earlier in Section II-A.

For our accelerator template, shown in Figure 3a, we take a 3×3 convolution and parallelize the inner convolution loops across a series of nine DSP48 blocks, four BRAMs, and two URAMs.The nine multiplications and eight additions are mapped to the nine DSPs in a sequential chain fashion. Each DSP48 computes the result of multiplying a weight with a corresponding input value and computes the partial sum of products. We operate the DSP in a SIMD=2 mode, i.e., we configure the 16b multiply and add datapath into two parallel 8b operations to enhance throughput. The DSP48 P cascade (Figure 2a) is used to accumulate the result of nine multiplication results.

We show snapshots of the internal DSP48 state in Figure 3b after cycle 7, 8 and 9. In the first six cycles, we initialize the pipelines from their empty state. We pipeline output of each multiply and add operation and orchestrate the input pixel shifting using a 2-stage pipeline to align data for systolic operation. The nine weights of a kernel are loaded into the B cascade chain of the DSP48 block and locked in place. The kernel BRAM store multiple sets of kernel weights that are accessed infrequently. The inputs are streamed over the A chain of the DSP48 blocks and split into three segments of length three. Using systolic pipeline mode of the DSP48 input chains, we are able to stream in a row and after an initial latency generate a stream of output pixels.

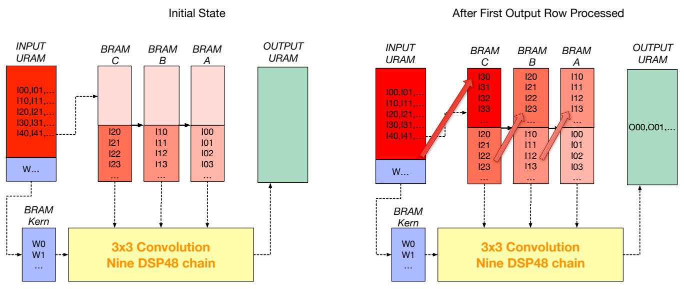


Fig. 4: Data movement between URAMs and BRAMs to support 3×3 convolution while exploiting row reuse. Row streaming over BRAM cascade overlapped with DSP compute.

Each segment is fed by a single BRAM thereby requiring three BRAMs to stream row pixels into the DSP48 chain. We achieve input row data reuse by copying the row data into the next BRAM as its been read out. Thus, in double-buffered fashion, the row data stages its way through the three BRAMs feeding into the three A segments. This is illustrated via snapshots of the row BRAMs in Figure 4. Three rows are fed in parallel to the Convolution Block while simultaneously being shifted into the next BRAM via the hard cascades. New rows are streamed from the URAM and one row is updated in the output URAM.

B. Building Block: Matrix-Vector Multiply

Unlike convolution, matrix-vector computations have low arithmetic intensity and require blocking to support diverging problem sizes across layers of the network.

Our accelerator design, shown in Figure 5, parallelizes the computation across a series of nine DSP48 blocks. The multiply-accumulate P cascade chain is identical to the one used in the convolution block with the exception of SIMD=1 mode configuration. Nine BRAMs feed 8b data to the adjacent DSP48 blocks to provide the vector input. The URAMs

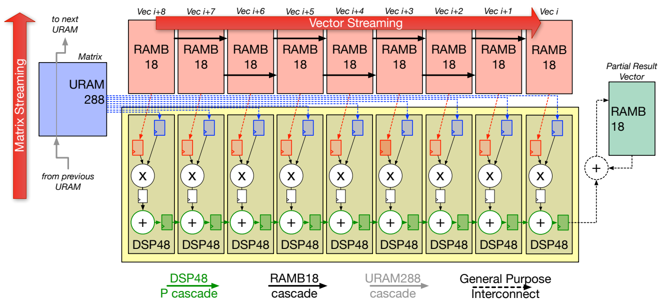


Fig. 5: Design of a Length-9 dot product unit that can perform URAM-capacity-limited matrix-vector multiplications. The DSP48 chain is configured with SIMD=1 mode to perform a matrix-vector product of 8b inputs. A chain of 9 DSPs is configured to perform a length-9 dot products. URAM distributes 9 chunks of 8b values from the matrix in a row-wise fashion. The bank of 9 BRAMs distributes 8b values.

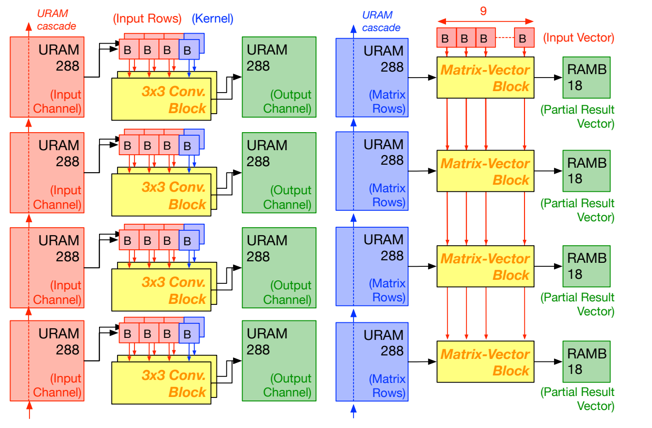
support 72b port widths, which are sliced across the nine DSP48s to distribute the matrix entries. The final result is accumulated in the result BRAM. The result is distributed across the FPGA via BRAM cascades so the output vector can be fanned out in preparation for the next computation.

C. Scaling and Tiling

The final FPGA organization that uses these building blocks for tiling must consider the unique DSP-BRAM-URAM capacity and bandwidth balance available on the UltraScale+ device family. We must also consider the deep network architecture as layer connectivity must be considered to ensure data movement between layers in handled properly. The Xilinx VU37P FPGA enforces a resource balance of 1 URAM288, 4.2 BRAM18s, and 9.4 DSP48s. Additionally the URAM can supply 72b data, while the RAMB18s can supply 18b data in True Dual Port (TDP) mode. The DSP48 blocks can consume inputs to feed the 27×18b multiplier and can be configured in SIMD=2 mode to compute two 8×8 multiplications with a common input.

We design repeating tiles to satisfy the resource-balance, bandwidth, and precision constraints of the FPGA device:

- **The Convolution Tile**: As shown in Figure 6a, each URAM supplies input channel data to two convolution blocks instead of one. The URAM has enough bandwidth to satisfy the needs of both blocks and helps create a repeating tile with 2 URAMs, 8 BRAMs, and 18 DSPs which is well within the VU37P balance of 2 URAMs, 8.4 BRAMs, and 18.8 DSPs. In addition, the URAM read/write cascades are employed to move the input channels across the different convolution tiles. This is necessary to support the all-to-all communication pattern inherent in an implementation of a convolution layer – here, each input channel convolves with a unique kernel for each output channel combination. We stream the input channel into a convolution block to update a particular output channel which is resident in the output URAM of that block. Simultaneously, the input channel is the shifted into the next URAM for the next output channel computation using the dedicated URAM cascade wiring. This all-to-all pattern is thus implemented by shifting data along a ring configured out of the URAM cascade structures.
- **Matrix-Vector Multiplication Tile**: In Figure 6b, we see that, we again need to generate a resource, bandwidth, and precision aware repeating tile. In this scenario, each URAM with its 72b port distributes matrix data to nine DSP48s in 8b chunks. We configure nine BRAMs to supply 8b vector data in parallel to the nine DSP48 blocks. Since the vector is common across all dot product evaluations, we fanout each BRAM output to the different copies of the Matrix-Vector multiplication blocks. Thus, each repeating tile has 4 URAMs, 9 BRAMs (input) + 4 BRAMs (output), 36 DSP48s which is well within the VU37P resource balance of 4 URAMs, 16.8 BRAMs, and 37.6 DSPs. If multiple FC layers are to be sequenced together, the resulting partial vector outputs stored in the output BRAMs are then shifted in a ring-like fashion across the multiple tiles to replicate the output vector across all tiles. This will the allow the next FC layer computation to proceed in an identical fashion.



(a) 3×3 Convolution Tiles (b) Matrix-Vector Multiply Tiles

Fig. 6: Repeating tiles of the ML accelerator that obey Xilinx VU37P resource, capacity, bandwidth constraints: Two Convolution tiles sharing the weight memory, while 4 tiles of the Matrix-Vector multiplication block share the vector RAM.

D. System Design Strategy

Finally, we determine the use of Space-Division multiplexing as the high-level parallelization strategy for supporting the different layer configurations on the same FPGA. We can partition the FPGA statically into two regions: one for convolutions, and another for matrix-vector multiplication. We can calibrate the balance based on the specific requirements of the deep network architecture (GoogLeNet splits across 80% convolution, and 20% matrix multiplication). To limit resource idling at the cost of inference latency, we (1) replicate the design to evaluate multiple images in parallel, or (2) decompose the FPGA into subregions devoted to a subset of layers of a CNN. Unlike the Xilinx SuperTile [29], [30] overlay, our design generalizes to a range of benchmarks beyond GoogLeNet. Reconfiguration was ruled out due to an exorbitant 140 ms of programming time [31].

E. Overall FPGA Architecture

The Xilinx VU37P FPGA supports HBM interfaces with 32× AXI ports with 256b 450 MHz rates feeding into the FPGA core. This is used for initial loading of the on-chip memory contents and is not needed therafter for all benchmarks (except *Transformer*, see Table I later) as the weights and worst-case activation state is <35 MB. . It also includes 960 URAM288 blocks, 9024 DSP48 slices, and 4032 RAMB18k blocks. Our architecture can support 960 computing blocks configured as 480 3×3 Convolution tiles, or 240 Length-9 Matrix-Vector Multiplication tiles. The Convolution tiles perform 2 3×3 convolutions across 18 DSP48 blocks configured in SIMD=2 mode thereby processing 36 8b×8b multiplications and 36 24b accumulations per cycle. The Matrix-Vector Multiplication tile can process four dot products of length 9 to yield a throughput of 36 8b×8b multiplications and 36 48b accumulations per cycle.

IV. Methodology

A. FPGA Mapping

We describe our designs directly in RTL component-level instantiations of DSP48, RAMB18, and URAM288 blocks and associated controllers for orchestrating data movement and control flow for convolutions and matrix-vector multiplication. We use Vivado 2018.2 for our experiments and use a tight 1 ns timing constraint for the CAD tools. We generate explicit physical location mapping for the DSP, and RAM components, and supply customizable pipelining to enforce the high-frequency design constraint. We measure the resulting frequency of the mapped design, and interconnect utilization metrics to quantify the extent of wiring reduction. We map our designs to the Xilinx UltraScale+ VU37P FPGA `xcvu37p-3`.

B. Performance Analysis of MLPerf Benchmarks

We build a cycle-accurate model of our design using an open-source CNN accelerator simulator from ARM called SCALE-Sim [20]. We model our system as a 9×1920 systolic array for Convolution and 9×960 array for Matrix-Vector Multiplication. Each grouping of 9 DSP48s in each chain are modeled as 1D systolic chains. We are able to model a variety of dataflows including the “weight-stationary” and “input-stationary” models for our design. For convolutions, our modeling framework includes support for parallelization of partial sum generation for an output channel. This feature can be added to our RTL design with a minor modification requiring an adder chain across multiple convolution blocks.

We run DNNs from MLPerf [1] to evaluate the performance of our cascade design along with GoogLeNet. We validate the cycle counts for various layers in GoogLeNet against that reported by our RTL simulation and SCALE-Sim runs. In Table I, we tabulate the peak memory usage footprint of MLPerf workloads that includes sum of all weights (filters and matrices) as well the worst-case activation layer storage costs. With the exception of *Transformer* benchmark, we never exceed the 35 MB capacity of the 960 URAMs on the VU37P.

TABLE I: MLPerf and GoogLeNet benchmark characteristics.

Topology	**Operation Count**			**Storage** (bytes)	
(MLPerf)	All	Conv	MM	$\sum$Wts.	Activ.
AlphaGoZero	352M	352M	353K	1.5M	92K
DeepSpeech2	1.7G	1.7G	74K	355K	6.5M
FasterRCNN	3.5G	1.6G	1.8G	13M	802K
NCF	11M	0	11M	11M	138K
Resnet50	3.4G	1.6G	1.8G	25M	802K
Sentimental	210M	0	210M	172K	30.7M
GoogLeNet	1.3G	1.3G	46M	6.8M	200K

V. Evaluation

We now discuss the implementation results of our interconnect-aware mapping of ML problems on Xilinx UltraScale+ VU37P FPGA. We first highlight the frequency and utilization of our proposed cascade design against one where the data movement is directly mapped over the soft fabric instead along with related work. We then discuss performance results for the MLPerf benchmark set for our platform and use GoogLeNet benchmark for comparison against Xilinx SuperTile.

A. Frequency Trends

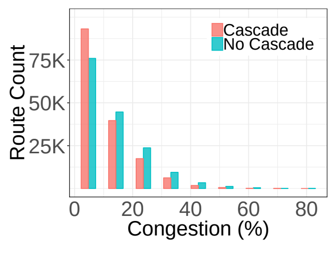
In Table II, we show the LUT and FF cost of the various design configurations along with frequency and interconnect utilization data. As expected, our careful bottom-up design methodology delivers high performance outcomes with the worst clock period of ≈ 1.5 ns. When considering the use of cascaded interconnect structures we observe a 15% reduction in clock period and 20–30% reduction in FF use. Convolution designs do not show much clock frequency improvements as our designs are extensively pipelined even without cascading features. A key measurement is the interconnect utilization drop of 16–33%. This is directly attributable to the use of cascade rather than general purpose interconnect for data movement.

We show the effect of cascading on network congestion through the histogram plots in Figure 7. It is clear that the cascaded design uses fewer congested routes than the non-cascaded design. This gap is stark for Convolution design with as many as 20% more routes in the lowest congestion bin. At the far end of the spectrum in the highly congested bins, we have 5–10× fewer routes for the cascaded design configuration.

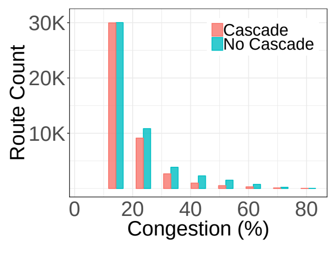
We show chip-spanning FPGA layouts of our Convolution and Matrix-Vector Multiplication designs in Figure 8. The regularity of the connectivity, and the use of nearest-neighbour cascade resources are visible in the layout. Complete FPGA mapping takes ≈6–7 hours on Vivado 2018.2 for these designs and is able to get close to the timing target of a ns. This easily outperforms the 250 MHz operating frequency of Brainwave design. Our implementation frequency is limited purely by

Design	Size	LUTs			FFs			Clk (ns)			Net Util. (%)		
		Fabric	Cascade	%	Fabric	Cascade	%	Fabric	Cascade	%	Fabric	Cascade	%
Convolution	Block	325	327	0%	1.3K	1K	30%	0.9	0.9	0%	0.01	0.01	0%
	Tile (2 blocks)	424	435	-2%	1.9K	1.5K	26%	0.9	1	-10%	0.02	0.02	0%
	Full-Chip	20.9K	21.1K	-1%	95.3K	72.1K	32%	1.4	1.4	0%	12.8	9.6	33%
Matrix-Vector Multiplication	Block	98	98	0%	775	688	12%	1	0.9	10%	0.01	0.01	0%
	Tile (4 blocks)	375	374	0%	2.3K	1.9K	21%	1.1	0.9	22%	0.04	0.05	-8%
	Full-Chip	90.2K	90.2K	0%	56.8K	46.6K	21%	1.5	1.3	15%	9.3	8	16%

TABLE II: Resource and Frequency Trends for Convolution and Matrix Vector Multiplication blocks, tiles and full-chip layouts.



(a) Convolution Congestion



(b) Matrix-Vector Congestion

Fig. 7: Histogram of congestion of routes for Full-Chip Convolution and Matrix-Vector Multiplication Hardware

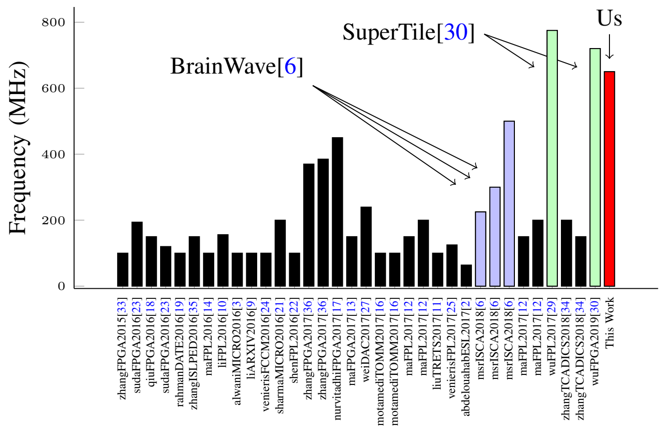
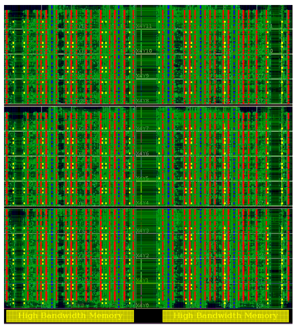


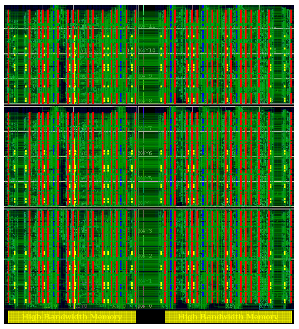
Fig. 9: Maximum Achieved Frequency of FPGA ML Accelerators over the past few years. Xilinx SuperTile [30] design has the highest 720 MHz fmax, and we operate at 650 MHz.

TABLE III: Xilinx VU37P FPGA inference latency (ms) and throughput (inf/s) for MLPerf benchmarks and GoogLeNet.

Topology (MLPerf)	**Ratio** (Conv:MM)	**Cycles**	**Time** (ms)	**Tput.** (inference/s)
AlphaGoZero	90:10	60K	0.09	10K
DeepSpeech2	60:40	1.2M	1.89	528
FasterRCNN	30:70	903K	1.38	719
NCF	0:100	2.4K	0.003	260K
Resnet50	30:70	848K	1.3	766
Sentimental	100:0	24K	0.037	27K
GoogLeNet (Us)	70:30	261K	0.40	2.4K
GoogLeNet (SuperTile [30])	-	-	3.3	3K



(a) Convolution Layout



(b) Matrix-Vector Layout

Fig. 8: Full chip VU37P layout of Convolution and Matrix-Vector Multiplication Hardware

the hard resource constraints than our design architecture. We ran an experiment by removing the URAM operations from our netlist and found the peak frequency achievable is 800–900 MHz in agreement with the limits of the DSP and BRAM components.

When compared to related work, our 650 MHz clock is within 70 MHz of the state-of-the-art 720 MHz Xilinx SuperTile [29], [30] design, and much faster than other contemporary designs as shown in Figure 9. While Xilinx SuperTile only uses 56% of the DSP48 blocks, we use 95% of our DSP48 resources delivering an effective throughput that is $\frac{100\%}{56\%} \times \frac{650\,MHz}{720\,MHz}$=1.6× better. Brainwave [6] operates between 225–500 MHz on the Stratix V–Stratix 10 silicon and is constrained by the memory controller interfacing speeds. In contrast, we operate like SuperTile by keeping weights and activations fully onchip in URAM288s and loading only once at the start.

B. Performance Trends

First, we tabulate the inference latency and throughput results in Table III. We see runtimes <2 ms across all benchmarks in the MLPerf set. In particular, we highlight the runtime of GoogLeNet at 0.4 ms which outperforms the 3.3 ms latency of the Xilinx SuperTile [30] design on the VCU1525 board with the VU9P-2 FPGA card (≈30% fewer DSP48s and 10% more BRAMs and identical URAM counts compared to VU37P FPGA). Our design uses almost all the DPS48s rather than the 56% of the SuperTile design, and also operates everything at the 650 MHz identical clock rather than half-rate RAM speeds of SuperTile. When considering throughput, the SuperTile array offers an impressive 3K images/sec of processing capacity. Our design can deliver a peak throughput

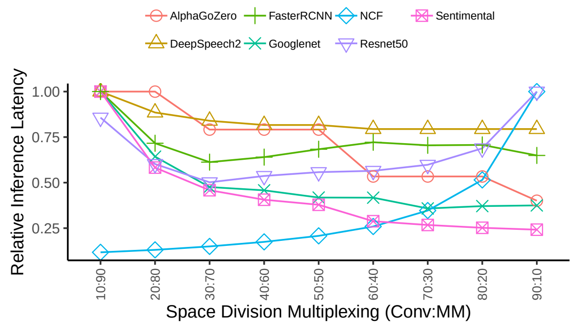


Fig. 10: Optimizing resource allocation for MLPerf workloads.

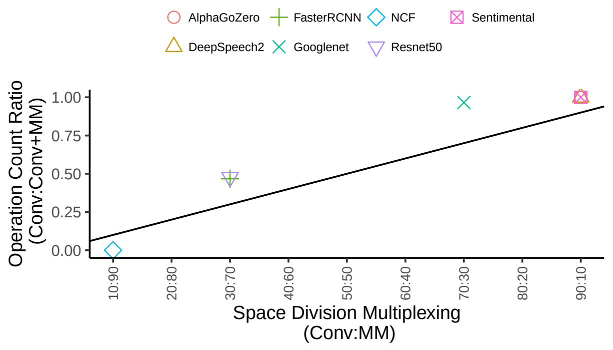


Fig. 11: Correlating MLPerf benchmark characteristics to Space Division Multiplexing arrangement.

of 2.4K images/sec which is 25% lower than SuperTile. The Xilinx design maximizes device utilization by pipelining execution across layers with three identical copies of the design, each with a chain of four processors sized differently and working on subset of the DNN layers. This sacrifices latency but improves throughput by allowing each processor to maximize device utilization. In contrast to the 95% utilization achieved by SuperTile, we only achieve 50% utilization but deliver superior inference latency.

In Figure 10, we show the benefits of systematic resource allocation of the FPGA to the different layers of the neural network. We divide resources to convolution:matrix-vector portions (the x-axis ratio shown in Figure 10) of the application keeping overall FPGA design area at 100%. Our current goal is to optimize for inference latency, and the particular balance of resources sacrifices some throughput to deliver superior inference latency outcomes. The resource balance at the runtime minimum point matches the ratio of work performed in the Convolution and Matrix-Vector Multiplication phases as indicated in Table I. *Resnet50* and *FasterRCNN* workloads shows a tradeoff that suggests best performance in the 30:70 resource division ratio. Rest of the workloads end up preferring a solution that is on either ends of the resource balance scale.

Finally, in Figure 11, we show a strong correlation between the problem requirements of the MLPerf benchmarks and the division of hardware resources to Convolution or Matrix-Vector multiplication tiles. Our optimization shows that there is a clean transition between the use of Convolution to Matrix-Vector multiplication hardware allowing us to stream the images through the chip.

VI. Lessons

Based on our study, we identify the following suggestions for future ML-friendly FPGA designs:

1. **URAM Bandwidth Balance**: A $2\times$ improvement in URAM memory bandwidth from each URAM will help address the memory bandwidth bottleneck for the matrix-vector multiplication phase of the computation.
2. **Dynamic Programmability**: For the Xilinx DSP48s, the cascades on the AB inputs, fracturing modes, and URAM cascades remain stubbornly statically configurable. This forces a designed to lock down data movement patterns at compile time and tailored uniquely for either convolution or matrix-vector multiply phases which makes a unified full-chip design is difficult.
3. **Impact of Xilinx Versal FPGA**: As stated earlier, we propose a closer look at existing hard interconnect structures within the FPGA fabric rather than embracing rigid ASIC-like computing elements targeting only the AI application domain. This departure also imposes a high design cost on the developer through the adoption of a mixed RTL and C/C++ or VLIW-assembly programming. The Versal system-level hard NoC does not address the intra-accelerator data movement requirements of computing workloads.

VII. Conclusions

This paper demonstrates the potential of mapping DNN data movements over cascade interconnect on the Xilinx UltraScale+ FPGA family. Our high-performance design can operate at a URAM-limited 650 MHz which compares favorably to the 720 MHz Xilinx SuperTile array, while outperforming it by $1.6\times$ on raw DSP48 throughput. For GoogLeNet we deliver $\approx 7\times$ superior inference latency but 30% lower inference throughput over the SuperTile design. MLPerf benchmarks operate at inference latencies of 3μs–1.89 ms and throughputs of 528–260K inf/s. Our design uses 30% fewer registers to get up to 12% faster clock frequencies while requiring 10–30% less interconnect than designs ignoring the cascade feature.

Source Code:
https://git.uwaterloo.ca/watcag-public/fpga-cascades-scalesim
https://git.uwaterloo.ca/watcag-public/fpga-cascades-rtl

References

[1] Mlperf: Mlperf benchmark suite. https://github.com/mlperf, 2018.
[2] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Sérot, Cédric Bourrasset, and François Berry. Tactics to directly map cnn graphs on embedded fpgas. *IEEE Embedded Systems Letters*, 9(4):113–116, 2017.
[3] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. Fused-layer cnn accelerators. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, page 22. IEEE Press, 2016.
[4] A. Boutros, S. Yazdanshenas, and V. Betz. Embracing diversity: Enhanced dsp blocks for low-precision deep learning on fpgas. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 35–357, Aug 2018.
[5] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *International Symposium on Computer Architecture (ISCA)*, 2016.

[6] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. A configurable cloud-scale dnn processor for real-time ai. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ISCA '18, pages 1–14, Piscataway, NJ, USA, 2018. IEEE Press.

[7] Yao Fu, Ephrem Wu, Ashish Sirasao, Sedny Attia, Kamran Khan, and Ralph Wittig. Deep learning with int8 optimization on xilinx devices. *Xilinx Whitepaper*, 2016.

[8] N. Kapre. Implementing fpga overlay nocs using the xilinx ultrascale memory cascades. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 40–47, April 2017.

[9] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[10] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, and Lingli Wang. A high performance fpga-based accelerator for large-scale convolutional neural networks. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–9. IEEE, 2016.

[11] Zhiqiang Liu, Yong Dou, Jingfei Jiang, Jinwei Xu, Shijie Li, Yongmei Zhou, and Yingnan Xu. Throughput-optimized fpga accelerator for deep convolutional neural networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 10(3):17, 2017.

[12] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. An automatic rtl compiler for high-throughput fpga implementation of diverse deep convolutional neural networks. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE, 2017.

[13] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 45–54. ACM, 2017.

[14] Yufei Ma, Naveen Suda, Yu Cao, Jae-sun Seo, and Sarma Vrudhula. Scalable and modularized rtl compilation of convolutional neural networks onto fpga. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE, 2016.

[15] Nick Mehta. Pushing performance and integration with the ultrascale+ portfolio. *Xilinx Whitepaper*, 2015.

[16] Mohammad Motamedi, Philipp Gysel, and Soheil Ghiasi. Placid: a platform for fpga-based accelerator creation for dcnns. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 13(4):62, 2017.

[17] Eriko Nurvitadhi, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason Ong Gee Hock, Yeong Tat Liew, Krishnan Srivatsan, Duncan Moss, Suchit Subhaschandra, et al. Can fpgas beat gpus in accelerating next-generation deep neural networks? In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 5–14. ACM, 2017.

[18] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 26–35. ACM, 2016.

[19] Atul Rahman, Jongeun Lee, and Kiyoung Choi. Efficient fpga acceleration of convolutional neural networks using logical-3d compute array. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1393–1398. IEEE, 2016.

[20] Ananda Samajdar, Yuhao Zhu, Paul N. Whatmough, Matthew Mattina, and Tushar Krishna. Scale-sim: Systolic CNN accelerator. *CoRR*, abs/1811.02883, 2018.

[21] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. From high-level deep neural models to fpgas. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, page 17. IEEE Press, 2016.

[22] Yongming Shen, Michael Ferdman, and Peter Milder. Overcoming resource underutilization in spatial cnn accelerators. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4. IEEE, 2016.

[23] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 16–25. ACM, 2016.

[24] Stylianos I Venieris and Christos-Savvas Bouganis. fpgaconvnet: A framework for mapping convolutional neural networks on fpgas. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 40–47. IEEE, 2016.

[25] Stylianos I Venieris and Christos-Savvas Bouganis. Latency-driven design for fpga-based convolutional neural networks. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE, 2017.

[26] Pete Warden. Why gemm is at the heart of deep learning. *Peter Warden's Blog*, 2015.

[27] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. Automated systolic array architecture synthesis for high throughput cnn inference on fpgas. In *Proceedings of the 54th Annual Design Automation Conference 2017*, page 29. ACM, 2017.

[28] Mike Wissolik, Darren Zacher, Anthony Torza, and Brandon Da. Virtex ultrascale+ hbm fpga: A revolutionary increase in memory performance. *Xilinx Whitepaper*, 2017.

[29] E. Wu, X. Zhang, D. Berman, and I. Cho. A high-throughput reconfigurable processing array for neural networks. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4, Sep. 2017.

[30] Ephrem Wu, Xiaoqian Zhang, David Berman, Inkeun Cho, and John Thendean. Compute-efficient neural-network acceleration. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2019, Seaside, CA, USA, February 24-26, 2019*, pages 191–200, 2019.

[31] Xilinx. Ug570: Ultrascale architecture configuration user guide. *Xilinx Whitepaper*, 2018.

[32] Xilinx. Versal: The first adaptive compute acceleration platform (acap). *Xilinx Whitepaper*, 2018.

[33] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 161–170. ACM, 2015.

[34] Chen Zhang, Guangyu Sun, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[35] Chen Zhang, Di Wu, Jiayu Sun, Guangyu Sun, Guojie Luo, and Jason Cong. Energy-efficient cnn implementation on a deeply pipelined fpga cluster. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pages 326–331. ACM, 2016.

[36] Jialiang Zhang and Jing Li. Improving the performance of opencl-based fpga accelerator for convolutional neural network. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 25–34. ACM, 2017.