DnnWeaver: From High-Level Deep Network Models to FPGA Acceleration

Hardik Sharma Jongse Park Emmanuel Amaro Bradley Thwaites
Praneetha Kotha Anmol Gupta Joon Kyung Kim
Asit Mishra[†] Hadi Esmaeilzadeh
Alternative Computing Technologies (ACT) Lab
School of Computer Science, Georgia Institute of Technology
[†]Intel Corporation
{hsharma, jspark, amaro, bthwaites, praneethak, anmol.gupta, jkkim}@gatech.edu
asit.k.mishra@intel.com hadi@cc.gatech.edu

Abstract

Deep Neural Networks (DNNs) are compute-intensive learning models with growing applicability in a wide range of domains. FPGAs are a compelling alternative to accelerate deep networks. However, using FPGAs for DNN acceleration is challenging since it requires long design cycles and expertise in hardware design. This work takes on this challenge, but instead of designing just an accelerator for a particular DNN model, it devises DnnWeaver*; a framework that* automatically *generates a synthesizable accelerator for a given (DNN, FPGA) pair using hand-optimized templates.* DnnWeaver *uses Caffe [1] as the programming interface to provide a high level of abstraction to the programmers. We use* DnnWeaver *to generate accelerators for the combinations of five different deep networks and two different FPGAs, Xilinx Zynq and Altera Stratix V. We rigorously compare the generated accelerators to multicore CPUs (ARM Cortex A15 and Xeon E3) and many-core GPUs (Tegra K1, GTX 650 Ti, and Tesla K40). In comparison, the generated accelerators deliver superior performance compared to CPUs and superior efficiency compared to GPUs; without requiring the programmers to participate in the arduous task of hardware design.*

1. Introduction

Deep Neural Networks (DNNs) are rapidly gaining traction in a wide range of applications [2, 3, 4, 5, 6, 7]. While their demand is growing, the benefits from general-purpose platforms are diminishing [8, 9]. In the light of these trends, the research community is increasingly turning to specialized accelerators [10, 11, 12, 13]. ASIC accelerators provide significant gains in performance and efficiency for deep networks [10, 11] at the price of high non-recurring engineering costs over long design periods. FPGAs are an attractive alternative and provide an intermediate point between the efficiency of ASICs and the generality of conventional processors. Nonetheless, FPGAs still require extensive hardware expertise and long design cycles. In fact, several inspiring research works [12, 13, 14, 15] have made extensive efforts to provide FPGA accelerators for specific DNN models, usually tailored for a particular FPGA platform. However, there is a lack of comprehensive and automated solutions to make FPGAs available to a broader community of DNN application developers who use a wide range of DNN models, and often lack any hardware design expertise.

This work seeks to provide such a solution by developing DnnWeaver, a comprehensive framework that generates synthesizable accelerators for a variety of FPGA platforms. The key challenge in developing DnnWeaver is completely disengaging the programmers from hardware design while providing significant performance and efficiency gains with FPGAs. This paper addresses this challenge and makes the following contributions:

(1) We develop a novel macro-dataflow *virtual* machine for DNN accelerators. The ISA for this virtual machine enables DnnWeaver to expose a high-level programming interface, and target different FPGAs and employ hardware optimizations without exposing them to the software.
(2) Instead of just designing an accelerator for DNNs, we develop hand-optimized template designs that are scalable and highly customizable. The templates provide a clustered hierarchical architecture that is shrunk or expanded by DnnWeaver to match the needs of a given (DNN, FPGA) pair.
(3) We develop a comprehensive compilation workflow that takes in the virtual ISA and generates the static execution schedule of the DNN accelerator as state machines and microcodes for the generated accelerator.

We use DnnWeaver to generate accelerators for five different deep networks and two different FPGAs, Xilinx Zynq and Altera Stratix V. We rigorously compare the generated accelerators to both multicore CPUs (ARM Cortex A15 and Xeon E3) and many-core GPUs (Tegra K1, GTX 650 Ti, and Tesla K40). The results are summarized as follows:

(1) On (Zynq, Stratix V), the DnnWeaver-generated accelerators provide ($7.6\times$, $43\times$) and ($1.3\times$, $7.1\times$) average speedup over ARM and Xeon, respectively.
(2) The generated accelerators for (Zynq, Stratix V) deliver ($2.2\times$, $1.2\times$), ($3.6\times$, $2\times$), and ($2.8\times$, $1.6\times$) higher Performance-per-Watt when compared to Tegra, GTX 650, and Tesla, respectively.

These results show that DnnWeaver takes an effective step towards making FPGAs available to DNN application developers without involving them in hardware design.

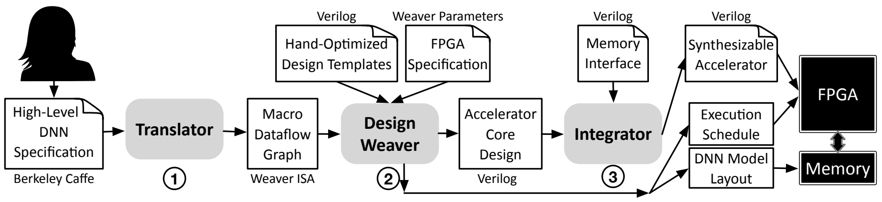


Figure 1: Overview of DnnWeaver which takes in high-level specification of a DNN and target FPGA and generates the accelerator design as synthesizable Verilog along with the accelerator execution schedule and the layout of DNN model in memory.

2. Overview of DnnWeaver

This work seeks to devise a comprehensive solution for utilizing a variety of FPGAs to accelerate a large class of DNNs. The objective is to offer a solution that (1) completely disassociates programmers from the details of hardware design and optimization while (2) providing a scalable and reusable FPGA acceleration platform, which delivers high performance and large efficiency gains for continuously changing DNN models on a variety of FPGA platforms. To achieve these two conflicting objectives, we developed DnnWeaver which combines hand-optimized scalable *template* designs with an *automated workflow* that customizes the templates to match a given (DNN, FPGA) pair of specifications. Figure 1 illustrates the DnnWeaver automated workflow comprised of three software components; (1) the Translator, (2) the Design Weaver, and (3) the Integrator. This section overviews these components after describing the programming interface.

Programming interface. The input to DnnWeaver is a high-level specification of the DNN in Berkeley Caffe format [1]. Caffe is a widely used open-source deep learning framework that takes the DNN specification as input and computes the given model on the CPU and GPU platforms. Using Caffe as the programming interface for DnnWeaver enables programmers to immediately transfer their DNNs from software to FPGA with no additional effort.

① **Translator.** The first component of DnnWeaver translates the Caffe specification of a DNN to our coarse-grained ISA that represents the DNN as a macro dataflow graph. We choose this coarse-grained abstraction to provide a unified hardware-software interface and enable layer-specific optimizations in the accelerator's microarchitecture without exposing them to the software. Furthermore, the ISA can be extended to support forthcoming layers or parameters. A one-time effort is required to develop the corresponding hardware templates. Section 3 describes the ISA.

② **Design Weaver.** By accepting the instructions representing the macro dataflow graph of the DNN, the Design Weaver generates a synthesizable Verilog implementation of the accelerator code using hand-optimized design templates. These templates provide a highly customizable, modular, and scalable implementation for the Design Weaver that automatically mutates and specializes the templates for the macro dataflow graph. A central aspect of this work is tuning the accelerator design to best utilize the available FPGA resources and achieve high performance and efficiency. Key FPGA resources include DSP slices (hard ALU blocks), on-chip Block RAMs (hard SRAM blocks), and the off-chip bandwidth. The Design Weaver shrinks or expands the accelerator structure depending on the availability of these resources in the target FPGA. Moreover, the Design Weaver also generates a static execution schedule for the generated accelerator as state machines and microcodes. Static scheduling simplifies the hardware and maximizes its efficiency and performance. DnnWeaver uses the proposed ISA to abstract away the accelerator as a *virtual dataflow machine* enabling static scheduling and avoiding the overheads of von Neumann execution (instruction fetch, decode, etc.). Additionally, as shown in Figure 1, the Design Weaver also generates the layout of the DNN parameters (weights) in the memory to streamline off-chip data fetch. Section 4 discusses the template designs in detail.

Bits	63 – 60	59 – 56	55 – 32	31	30 – 24	23 – 17	16	15 – 10	9 – 4	3 – 0
input	Opcode = 0	Function	Destination Instruction ID	Fixed-Point/ Floating Point	Value Bitwidth	Fraction Bits/ Exponent Bits	1	# of Dimensions		
conv	Opcode = 1						Use the Next Word	Window Width	Window Height	Window Stride
pool	Opcode = 2									
norm	Opcode = 3									
ip	Opcode = 4							# of Neurons		
act	Opcode = 5						0	Reserved		
fanout	Opcode = 6		# Destinations							
output	Opcode = 7		0							

Figure 2: Instructions of the macro-dataflow virtual machine.

③ **Integrator.** As depicted in Figure 1, the last component of DnnWeaver is the Integrator, which adds the memory interface code to the accelerator code. The Integrator contains a library of DRAM interfaces and adds the appropriate code for the target FPGA. After the integration, the final synthesizable Verilog code can be synthesized on the target FPGA to accelerate the specified DNN in Caffe format.

3. Instruction Set Architecture Design

DnnWeaver provides a coarse-grained *virtual* ISA to (1) abstract away the details of the accelerator design from the software; (2) enable layer-specific optimizations; (3) facilitate portability across different FPGA platforms; and (4) allow static execution scheduling at compile time. To achieve these goals, we design a *macro dataflow virtual machine* for DnnWeaver and expose its ISA to the software.

One of the main pillars of any accelerator design is to alleviate or minimize the von Neumann overhead of the general-purpose architectures that includes instruction fetch, decode, etc. To minimize this overhead, we chose a dataflow architecture as our virtual machine. This dataflow virtual

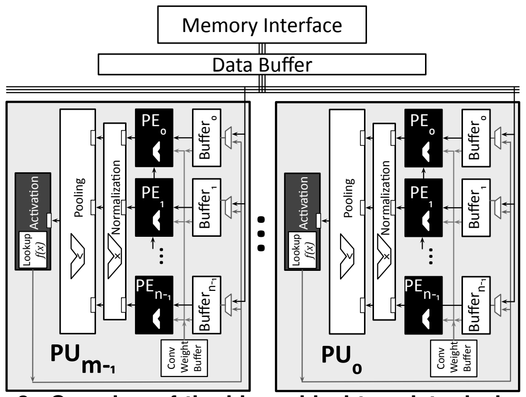


Figure 3: Overview of the hierarchical template design. The template accelerator is clustered and divided into Processing Units (PUs), comprised of multiple Processing Engines (PEs).

machine does not have explicit registers, which enables DnnWeaver to impose significantly fewer restrictions on the accelerator architecture and allows portability across different FPGAs. Being an explicit dataflow architecture also allows DnnWeaver to perform static optimizations at compile time and avoid data dependence (e.g., register renaming) at runtime. Additionally, the coarse grain nature of the ISA enables the microarchitecture to incorporate layer-specific optimizations without exposing them through the software stack. Figure 2 shows all the eight instructions of our virtual machine encoded using 64-bits. All these instructions are variable-sized and are designed to be able to express and implement a large variety of deep neural networks. None of the instructions of this dataflow architecture include source operands. Instead, a part of instruction opcode encodes the unique static ID of the destination instruction that will receive the results. To support this explicit data dependence between instructions, each instruction in the binary is assigned a unique 24-bit static ID.

The virtual instructions are translated to state machine and microcodes at compile time. Therefore, we generously allocated long words to the instructions to increase generality. All these instructions consume multiple input elements, perform complex operations, and produce multiple output elements. Therefore, our virtual machine for DNN acceleration is a *macro* dataflow machine. As we discuss in the next section, the Design Weaver utilizes this virtual ISA to generate the accelerator code that is best suited for a pair of (DNN, FPGA).

4. Template Accelerator Architecture

The Design Weaver uses hand-optimized predesigned templates and automatically generates an accelerator that is best tuned for a given DNN and a target FPGA. The template designs are highly *customizable* and *scalable*, which enables the Design Weaver to shrink or expand the accelerator based on the requirements of the DNN and the availability of the resources in the target FPGA. Generality is another aspect of the template design. That is, the templates include exchangeable hardware components that realize different layers of DNNs. If a DNN does not include a certain layer (e.g., normalization), the corresponding component is excluded to enhance resource utilization. Excluding these components can potentially provide more headroom to the rest of the necessary components in the accelerator, which can lead to higher performance.

Overall Organization Figure 3 illustrates the template architecture that provides these necessary characteristics. As depicted, the template architecture is clustered with two levels of hierarchy; a collection of self-contained Processing Units (PUs) that comprise a set of smaller Processing Engines (PEs). The PEs and the buffers in the template PU architecture provide compute capabilities for convolution and inner product layers. The customizable normalization, pooling, and activation modules provide support for the other possible layers in DNNs. This clustered architecture provides scalability not only via modularity but also by making the data traffic local to PUs and utilizing an untied bussing fabric across them. These features allow the Design Weaver to generate a concrete accelerator with any number of PUs and PEs-per-PU. Furthermore, each hardware module is extensively parameterized. The Design Weaver tunes these parameters based on the DNN topology, its layers, and the amount of resources in the FPGA.

Specializing the design for a target FPGA. Each FPGA offers a certain number of hard blocks including DSP slices (ALUs) and Block RAMs (on-chip SRAM units, called BRAMs). Making use of these hard blocks is essential for achieving reasonably high frequency with FPGAs. Thus, the template architecture in Figure 3 maps the PU Buffers to the BRAMs and the ALUs to the DSP slices. The availability of these resources determines the maximum possible number of PEs and PUs for a given FPGA. However, the Design Weaver determines the final composition of the PUs based on the DNN topology and layers. The composition of the PU is determined based on the size of the output feature maps produced by the convolution/pooling/normalization/IP layers to maximize resource utilization and also maximize overall computation throughput. The next resource is the available off-chip bandwidth which determines the parameters of the Data Buffer that is connected to the memory interface as shown in Figure 3. The Design Weaver performs static data marshaling and determines the layout of the DNN weights and parameters to streamline transfer parameters from the memory in contiguous chunks. The Design Weaver also generates a static schedule for the Data Buffer to fetch and feed the PUs through the inter-PU bus. Static scheduling avoids contention on the bus and alleviates the need for PUs to perform complex handshaking. This approach, in turn, improves the scalability and efficiency of the template architecture.

5. Evaluation

To evaluate the effectiveness of DnnWeaver, we use two off-the-shelf FPGA platforms, Xilinx Zynq ZC702 and Altera Stratix V GS D5. Table 1 summarizes their specifications. We investigate the performance and energy benefits of the generated accelerators compared to a diverse set of high-end and low-end CPUs and GPUs. For the non-FPGA results, we

Table 1: FPGA Platforms.

FPGA hardware platforms		
Model	Xilinx Zynq ZC702	Altera Stratix V SGSD5
FPGA Capacity	53K LUTs	172K LUTs
	106K Flip-Flops	690K Flip-Flops
Peak Frequency	250 MHz	800 MHz
BRAM	630 KB	5035 KB
MACC Count	220	1590
Price	$129	$6,995
Technology	TSMC 28nm	TSMC 28nm

Table 2: Benchmark DNNs and their input datasets. The topology column specifies the layers of DNN (C: Convolution, P: Pooling, I: Inner Product, A: Activation, N: Normalization).

Network	Data set	Domain	Topology
LeNet	MNIST	Handwritten Digit Recognition	C->P->C->P->I->A->I
Siamese	MNIST	Handwritten Digit Recognition	C->P->C->P->I->A->I->I
CIFAR-10 Full	CIFAR-10	Object Recognition	C->P->A->N->C->A->P->N->C->A->P->I
CifAR-10 Quick	CIFAR-10	Object Recognition	C->P->A->C->A->P->C->A->P->I->I
Djinn ASR	Kaldi	Speech-to-text decoder	I->A->I->A->I->A->I->A->I->A->I->A->I

Table 3: Evaluated CPUs and GPUs.

Platform	Cores	Clock freq (GHz)	Memory (GB)	TDP (W)	Technology (nm)	Cost
ARM Cortex A15	4+1	2.3	2 (shared)	5	28	$191
Intel Xeon E3-1246 v3	4	3.6	16	84	22	$290
Tegra K1 GPU	192	0.852	2 (shared)	5	28	$191
NVIDIA GTX650Ti	768	0.928	1	110	28	$150
Tesla K40	2880	0.875	12	235	28	$5,499

employ Berkeley Caffe. All platforms are benchmarked with five DNN models. Henceforth, we refer to DNNWEAVER-generated accelerator for Stratix V and Zynq as DW-Stratix and DW-Zynq, respectively.

5.1. Methodology

Benchmark DNNs and their Input Datasets. Table 2 shows our benchmark DNN models and their input datasets. Among the benchmarks, the LeNet and Siamese networks target the popular MNIST handwritten digit recognition dataset [16]. The CIFAR-10 networks target object detection in the CIFAR-10 thumbnail dataset [17]. The DjiNN ASR network is a DNN speech-to-text decoder obtained from the DjiNN and Tonic benchmark suite [7]. The network definitions for LeNet, Siamese, and CIFAR-10 are available in Caffe.

Runtime measurements. We compare the execution time of DNNWEAVER generated accelerators to the execution time on CPUs and GPUs using Berkeley Caffe. Table 3 lists the five evaluated CPUs and GPUs. The CPU and GPU baselines are compiled with GCC 4.8 and NVCC 6.5, respectively. We obtain the baseline timings by using the timing feature of Caffe. Across all the platforms, we run each DNN 100 times and use the average.

FPGA platforms details. In the Zynq board, the interface between DRAM and programmable logic is a standard AXI bus. In the Stratix board, we used Altera's Avalon interface IP. We implement a custom controller on the programmable logic to interface with the AXI and Avalon interfaces on the two boards and transfer data to and from the main memory. We synthesize the hardware with 64-bit Vivado v2015.1 for the Zynq board and Qartus II v14.1 for the Stratix board.

Power measurements using vendor libraries. We use the NVIDIA Management Library (NVML) to obtain the average power of Tesla K40. Given that GTX650Ti does not support the NVML library, and since the GTX650Ti and Tesla K40 share the same microarchitecture, we make a conservative estimation of the GTX650Ti power by scaling the Tesla K40 measurements using the two chips' TDPs. For each DNN, we calculated the ratio of measured power in Tesla K40 over its TDP. We multiply this ratio with the GTX650Ti TDP and use 95% of this number.

We use the Intel Running Average Power Limit (RAPL) energy consumption counters available in the Linux kernel for power measurements on the Xeon E3.

Power measurements in hardware. The ARM Cortex A15 CPU and the Tegra K1 GPU are a part of the Jetson TK1 development board. We use the Keysight E3649A Programmable DC Power Supply to get the power consumption of the complete Jetson TK1 board. To do so, we subtract the idle average power 3.12W from the power reading we obtain during benchmark execution. Similarly, we make use of a Tektronix 2280 DC Power Supply to measure the power of Stratix V. Finally, we utilize a GPIO to USB adapter to read the power directly from the power controllers in the Zynq board.

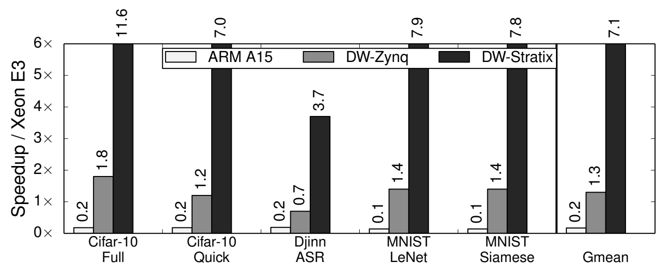
5.2. Experimental Results

To investigate the performance gains of the DNNWEAVER-generated accelerators, DW-Zynq and DW-Stratix, we perform an extensive comparison with diverse CPU and GPU platforms. We present these results separately.

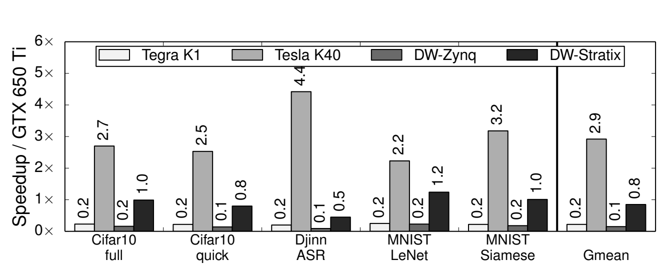
Performance comparison with CPUs. Figure 4a illustrates the performance benefits when DW-Zynq and DW-Stratix are used to compute the models under evaluation. The performance of Xeon E3 is used as a baseline for comparison. The average speedup for a pair of (DW-Zynq, DW-Stratix) is (1.3×, 7.1×); thus, DW-Stratix provides 5.6× more speedup than DW-Zynq. Among the evaluated models, Cifar-10 Full sees the highest speedup of (1.8×, 11.6×) while Djinn ASR shows the lowest speedup of (0.7×, 3.7×). The significant gap in performance benefits comes from the disparity in the model topology since some layers are more favorable to the DNNWEAVER-generated accelerators than the others. Compared to the low-end ARM processor, DW-Zynq and DW-Stratix provide 7.6× and 43× speedup respectively.

These results demonstrate the performance benefits provided by DNNWEAVER-generated accelerators over both low-end and high-end CPU platforms, as well as their scalability over various FPGA platforms.

Performance comparison with GPUs. We compare our accelerators with GPU platforms including GTX650Ti, Tegra K1, and Tesla K40 in Figure 4b. The baseline is GTX650Ti, a middle-tier GPU. DW-Zynq provides 0.15× performance compared to GTX650Ti, while DW-Stratix provides 0.85× performance. The maximum speedup of (0.16×, 1×) is observed from Cifar-10 Quick, whereas Djinn ASR shows the minimum speedup of (0.1×, 0.5×). The low-end GPU, Tegra K1, offers a 0.2× average speedup over the baseline. In contrast, the high-end GPU, Tesla K40, presents a 2.9× speedup. Compared to Tesla K40, DW-Zynq and DW-Stratix



(a) Speedup of ARM A15 and DnnWeaver-generated accelerators over Xeon E3.



(b) Speedup of Tegra K1, Tesla K40, and DnnWeaver-generated accelerators over GTX650Ti.

Figure 4: Speedup of DnnWeaver-generated accelerators in comparison to a range of CPU and GPU platforms.

show an average slowdown of 0.05× and 0.3×, respectively.

Performance-per-Watt comparison with CPUs. As shown in the speedup results, the performance benefits from diverse CPU, GPU, and FPGA platforms vary substantially. In fact, these hardware platforms occupy different design points in the underlying performance vs. energy efficiency tradeoff. To understand the performance benefits with the fixed energy efficiency, we measure the power consumption and evaluate the performance-per-Watt for each hardware platform.

Figure 5a delineates the comparison of performance-per-Watt for ARM A15, DW-Zynq, and DW-Stratix with the baseline of Xeon E3. On average, DW-Zynq shows 36× and DW-Stratix shows 20× higher performance-per-Watt than the baseline. Note that although DW-Stratix provides about 5.6× higher speedup, the increased power consumption by DW-Stratix (2W vs. 25W) leads to the lower performance-per-Watt than DW-Zynq. This trend is observed for all the evaluated DNN models.

Low-end processors such as ARM A15 are commonly used in mobile devices and are known to have high energy-efficiency. We also compare the ARM A15 processor with our accelerators and Xeon E3. The ARM A15 processor shows 2.3× higher performance-per-Watt compared to Xeon E3. When compared with ARM A15, the pair of (DW-Zynq, DW-Stratix) shows 15× and 8.7× higher performance-per-Watt, which demonstrates the energy efficiency of the DnnWeaver-generated accelerators.

Performance-per-Watt comparison with GPUs. Figure 5b shows the performance-per-Watt in comparison of Tegra K1, Tesla K40, DW-Zynq, and DW-Stratix with the baseline, GTX650Ti. The pair of (DW-Zynq, DW-Stratix) provides (3.6×, 2×) higher performance-per-Watt than the baseline. Although DW-Stratix outperforms DW-Zynq with the speedup of 5.6× shown in Figure 4b, DW-Zynq offers a 1.7× higher performance-per-Watt compared to DW-Stratix.

On average, Tegra K1 and Tesla K40 have 1.7× and 1.3× higher performance-per-Watt than GTX650Ti. Even though Tesla K40 provides 2.9× speedup in comparison with Tegra K1 shown in Figure 4b, Tegra K1 provides 30% higher performance-per-Watt, which indicates the high energy efficiency of the platform.

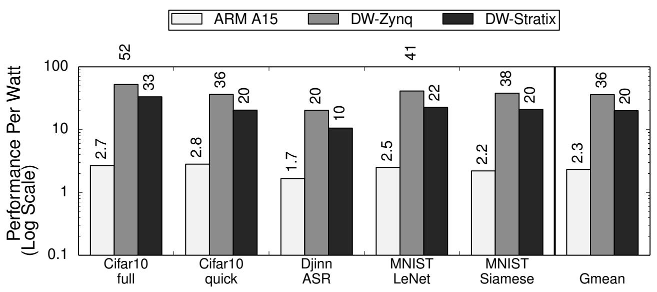
6. Related Work

There have been several proposed architectures that accelerate machine learning algorithms and DNNs. However, DnnWeaver fundamentally differs from these efforts since it is not just an accelerator, but an accelerator generator. DnnWeaver is capable of producing an optimized design for a given (DNN, FPGA) pair. DnnWeaver also provides a novel ISA and macro-dataflow virtual machine to unify DNN accelerators across different FPGA platforms. Below, we discuss the most related work in the area of FPGA implementations and ASIC accelerators for DNNs.

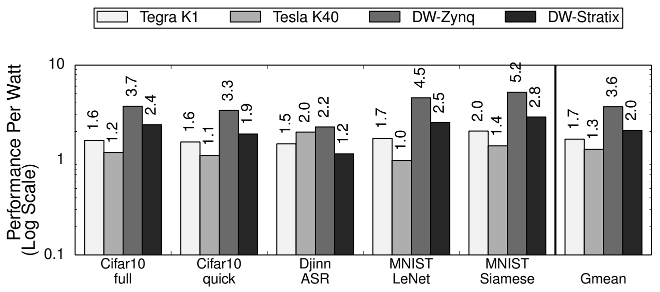
FPGA implementations for DNNs. The work by Chen, et al. [12] focuses on using an analytical design scheme based on the roofline model to find the fastest design for a particular DNN for FPGA acceleration. Their design does not support some DNN layers such as pooling and normalization – these layers are supported by DnnWeaver. The work by Farabet, et al. [13, 15] develops a specific FPGA accelerator for a specific DNN. Gokhale, et al. [14] develop a mobile co-processor for DNNs and evaluate it on a Zynq board. Chakradhar, et al. [18] develop a VLIW coprocessor for DNNs and *emulate* it on a Virtex 5 FPGA. They propose a special switch that allows to dynamically group the convolution engines in different ways. The design has a low-level VLIW ISA but the paper does not include any details about its design. Unlike DnnWeaver, they do not generate Verilog code for FPGA accelerators. In a press release, Microsoft stated they were working on extending the Catapult platform for deep learning. However, no details about the design are available for a meaningful comparison [19].

While prior work has explored various individual design points for accelerating deep learning, our work focuses on making FPGA accessible to a wide range of DNN developers by automatically generating an optimized accelerator from an abstract high level specification for the DNN and the target FPGA. In addition, these works differ from DnnWeaver as they do not generate accelerators, do not provide virtual ISAs or a virtual machine for DNN accelerators, or do not start from high level abstractions.

ASIC accelerators for DNNs. (Da)Diannao [10, 11] provides two low power deep learning ASICs. (Da)Diannao also provides a low-level fine-grained ISA, but unlike DnnWeaver, they do not define a virtual ISA or virtual machine to unify deep accelerators. Ngyen, et al. [20] propose using Coarse Grained Reconfigurable Architectures (CGRAs) to implement neural networks. Qadeer, et al. [21] develop a Convolution



(a) Performance-per-watt of CPUs and the DNNWEAVER-generated accelerators in comparison to Xeon E3.



(b) Performance-per-watt of GPUs and the DNNWEAVER-generated accelerators in comparison to GTX650Ti.

Figure 5: Performance-per-watt of the DNNWEAVER-generated accelerators in comparison to a range of CPU and GPU platforms.

Engine, which transforms convolution into a map and reduces operations which can be used for accelerating the convolution layer of the DNNs. Conti, et al. [22] develop convolution cores designed to integrate with a shared-memory cluster of RISC processors. PuDianNao [23] provides a generic ASIC accelerator that targets machine learning techniques in general, but excludes deep convolutional networks. These previous efforts require ASIC design and do not generate accelerators for FPGA realization, which is the focus of our work.

7. Conclusion

DNNs are compute-intensive workloads that can significantly benefit from acceleration. This paper, described DNNWEAVER, aims to close the gap between high-level specification of DNNs and FPGAs. DNNWEAVER provides a novel virtual ISA and a macro-dataflow virtual machine to automatically generate a concrete accelerator from a high-level specification for a given (DNN, FPGA) pair. While providing automation, DNNWEAVER delivers high performance and efficiency by generating the accelerator code from a series of scalable and customizable hand-optimized template designs. We rigorously compare the DNNWEAVER generated accelerators to five different CPUs and GPUs. In comparison, the generated accelerators deliver significantly superior performance compared to CPU platforms, and superior efficiency compared to GPU platforms; without requiring the programmers to participate in the arduous task of hardware design. These results suggest that DNNWEAVER takes an effective step in making FPGAs available to a broader community of DNN developers who do not often possess hardware expertise.

8. Acknowledgements

This work was supported in part by Microsoft, Qualcomm, Google, NSF awards CCF #1553192 and CNS #1526211, and Semiconductor Research Corporation contract #2015-TS-2636.

References

[1] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[2] J. Hauswald, M. A. Laurenzano, Y. Zhang, C. Li, A. Rovinski, A. Khurana, R. Dreslinski, T. Mudge, V. Petrucci, L. Tang, and J. Mars, "Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers," in *ASPLOS*, 2015.

[3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[4] G. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[5] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *ICML*, 2009.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[7] T. M. R. G. D. Jason and M. L. Tang, "Djinn and tonic: Dnn as a service and its implications for future warehouse scale computers," 2015.

[8] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *ISCA*, 2011.

[9] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Toward dark silicon in servers," *IEEE Micro*, vol. 31, pp. 6–15, July–Aug. 2011.

[10] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ASPLOS*, pp. 269–284, ACM, 2014.

[11] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, *et al.*, "Dadiannao: A machine-learning supercomputer," in *MICRO*, pp. 609–622, IEEE, 2014.

[12] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *FPGA*, pp. 161–170, ACM, 2015.

[13] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision," in *CVPRW*, 2011.

[14] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *CVPRW*, 2014.

[15] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *ISCAS*, 2010.

[16] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[17] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Computer Science Department, University of Toronto, Tech. Rep*, vol. 1, no. 4, p. 7, 2009.

[18] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 38, pp. 247–257, ACM, 2010.

[19] Microsoft, "Microsoft extends fpga reach from bing to deep learning." http://www.nextplatform.com/2015/08/27/microsoft-extends-fpga-reach-from-bing-to-deep-learning/, 2015.

[20] T. Ngyen, S. Jafri, M. Daneshtalab, A. Hemani, S. Dytckov, J. Plosila, and H. Tenhunen, "Fist: A framework to interleave spiking neural networks on cgras," in *PDP*, pp. 751–758, March 2015.

[21] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, "Convolution engine: balancing efficiency & flexibility in specialized computing," in *ACM SIGARCH Computer Architecture News*, vol. 41, pp. 24–35, ACM, 2013.

[22] F. Conti and L. Benini, "A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters," in *DATE*, 2015.

[23] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "Pudiannao: A polyvalent machine learning accelerator," in *ASPLOS*, pp. 369–381, ACM, 2015.