Flexibility: FPGAs and CAD in Deep Learning Acceleration

Gordon R. Chiu
Intel Corporation
gordon.chiu@intel.com

Andrew C. Ling
Intel Corporation
andrew.ling@intel.com

Davor Capalija
Intel Corporation
davor.capalija@intel.com

Andrew Bitar
Intel Corporation
andrew.bitar@intel.com

Mohamed S. Abdelfattah
Intel Corporation
mohamed.abdelfattah@intel.com

ABSTRACT

Deep learning inference has become the key workload to accelerate in our AI-powered world. FPGAs are an ideal platform for the acceleration of deep learning inference by combining low-latency performance, power-efficiency, and flexibility. This paper examines the flexibility aspect, and its impact on FPGA design methodology, physical design tools and CAD. We describe the degrees of flexibility required for creating efficient deep learning accelerators. We quantify the varying effects of precision, vectorization, and buffering on both performance and accuracy, and show how the FPGA can yield superior performance through architecture customization tuned for a specific neural network. We describe the need for abstraction and propose solutions in modern FPGA design flows to enable the rapid creation of these customized accelerator architectures for deep learning inference acceleration. Finally, we examine the implications on physical design tools and CAD.

KEYWORDS

Deep Learning; FPGAs; High-Level Design; Physical Design

ACM Reference Format:
Gordon R. Chiu, Andrew C. Ling, Davor Capalija, Andrew Bitar, and Mohamed S. Abdelfattah. 2018. Flexibility: FPGAs and CAD in Deep Learning Acceleration. In *ISPD '18: 2018 International Symposium on Physical Design, March 25–28, 2018, Monterey, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3177540.3177561

1 INTRODUCTION

Over the past few years, deep learning techniques have been increasingly applied to solve problems in many fields, starting with computer vision and speech recognition, but growing to include natural language processing, machine translation, autonomous vehicles and smart medicine. As of 2016, analytics was the fastest growing workload in the datacenter, and predicted to be the largest workload by compute cycles in 2020, mostly due to the rapid increase in adoption and deployment of deep learning. [4]

Unlike deep learning training, which is predominantly hosted in data-centers and the cloud, deep learning inference – the scoring of a trained neural network model against unknown input data – is often performed in-line with data collection, which could either be in a data-center context (for web-scale analytics, image or media processing) or in an embedded context (within autonomous driving systems or surveillance systems). In both contexts, system architects are looking to offload inference compute cycles from valuable CPUs, leveraging in-line or off-load compute accelerators, such as General-Purpose Graphics Processing Units (GPGPUs), FPGAs, or fixed-function ASICs. The enhanced data parallelism available in an accelerator can both improve power-efficiency as well as reduce compute latency. This reduced latency manifests in better system performance (such as response time to a user request over the web, or reaction time to external events in an autonomous driving system).

When data scientists look to solve a problem with deep learning, they typically begin with a "standard" neural-network topology (typically an ILSVRC [10]-winning network, such as GoogLeNet [9] or ResNet [13]) and iterate from there. With the rapid change in this space, the final network is often not defined at the beginning of the project, and can morph as compute, accuracy, and application requirements change. While general-purpose compute technologies such as CPUs are flexible enough to adapt to neural-network topology changes, the performance of an fixed-function accelerator varies profoundly with the choice of topology. This leads to a phase-ordering problem, as the acceleration hardware and design in a system is often locked down well before the workload is finalized, leading to subpar accelerator performance.

As research continues, the industry may consolidate on key standard network topologies. Until then, with new topologies and innovations emerging on a daily basis, flexibility in the accelerator is critical for supporting a wide gamut of network topologies. This flexibility has large implications on the physical design flows, tools, and CAD required to efficiently define and create accelerators.

1.1 Why FPGAs for Deep Learning Acceleration?

The FPGA architecture is naturally amenable for deep learning inference acceleration. Arithmetic datapaths can be synthesized and mapped to reconfigurable logic, for greater power-efficiency and lower latency than executing instructions temporally on a CPU or GPGPU. System integration options through flexible I/O interfaces allow tighter native integration into embedded or streaming applications (such as directly processing the output of a camera module). Most importantly, flexibility in the reconfigurable logic and routing enables many different accelerator architectures to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ISPD '18, March 25–28, 2018, Monterey, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5626-8/18/03…$15.00
https://doi.org/10.1145/3177540.3177561

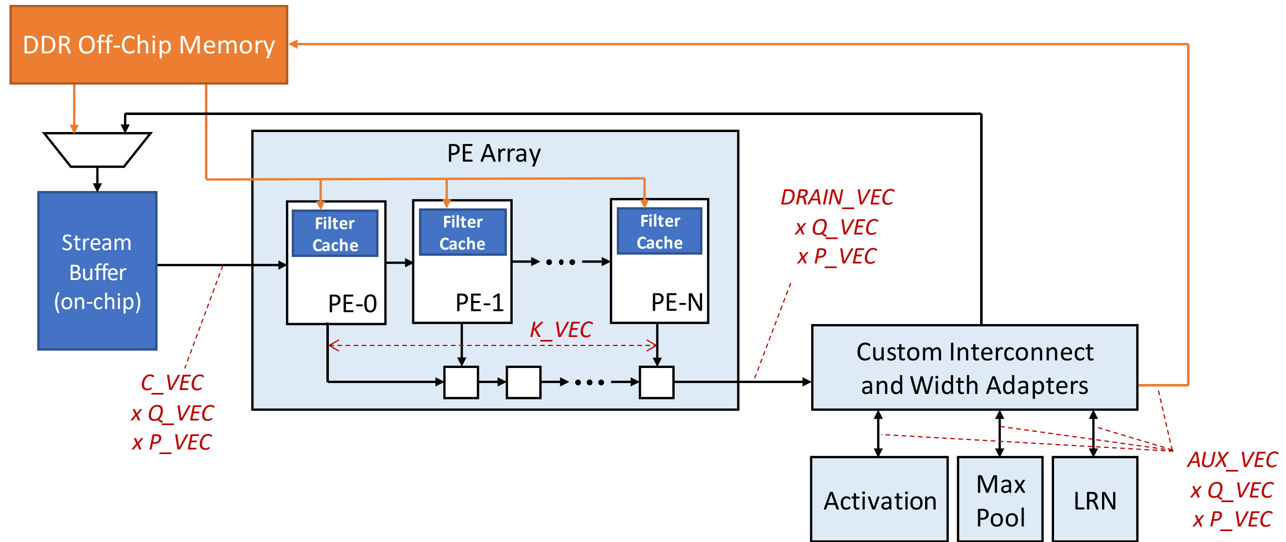

Figure 1: System-level diagram of our neural-network inference accelerator.

be efficiently implemented on the same FPGA fabric. This paper focuses on this flexibility argument for FPGAs, argues the need for higher-level design, and presents implications on CAD (both high-level and physical):

- **Flexibility inherent in the FPGA fabric** enables many accelerator architectures to be constructed. The optimization upside from using a bespoke acceleration architecture, tuned to a specific network, can outweigh the cost of the reconfigurable fabric of the FPGA. This exposes a very large design space.
- **High- and higher-level design solutions** are required to take full advantage of this architecture flexibility and large design space. Our current solution comprises of an acceleration stack, and a high-level synthesis compiler, providing good abstraction. Further abstraction is necessary for arbitrary accelerator generation.
- **New challenges for CAD and Physical Design** are exposed by the higher levels of design abstraction and the domain's high performance requirements. We describe some of our solutions to problems seen today, as well as motivate and propose areas of future research.

The remainder of the paper is organized as follows. Section 2 describes our Deep Learning Accelerator (DLA) architecture. Section 3 examines the various degrees of flexibility present when implementing an accelerator on an FPGA, and quantitatively analyzes the benefit of customizing accelerator to specific deep learning workloads (as opposed to a fixed-function accelerator). Section 4 describes our current high-level design abstraction for creating the Deep Learning Accelerator, and proposes future higher-levels to enable success in this space. Finally, Section 5 details some of the challenges in mapping generated designs to physical implementations, along with some workarounds employed as well as proposals for areas of future research.

2 FPGA DEEP LEARNING ACCELERATOR

In [2], we presented an architecture and implementation for high-performance on-FPGA execution of deep learning inference acceleration. The system implemented on-FPGA is summarized in Figure 1. An accelerator core reads input image and filter data from external (DDR) memory, and stores the data in caches built of on-chip block RAMs. Data is read from on-chip caches and fed to a set of parallel Processing Elements (PEs), which perform, in parallel, the dot product calculations that comprise the bulk of the workload. A drain network collects the output of the dot product, which is fed to *auxiliary kernels* that perform the non-linear computations (such as activation, pooling, and normalization). The resulting output is either fed as input to the next layer of convolution (executed sequentially), or written as an output back to external memory.

3 DEGREES OF FLEXIBILITY

Though our original work focused on a single optimized instance of an accelerator for the AlexNet neural network topology [2], our architecture is flexible enough for high-performance acceleration of other network topologies. The design can be parameterized to create a class of neural network accelerator designs that can be specialized for specific network topologies. In this section, we examine a few of the degrees of flexibility available, and quantify the benefit provided by the FPGA flexibility, across some example axes of compute precision, accelerator geometry, and memory architecture.

3.1 Compute Precision

Deep Learning applications often have large memory requirements, which pose a hurdle to their acceleration. Recent work has shown that reducing the accelerator's precision from full-precision floating-point can shrink the neural network model size while maintaining its required accuracy.

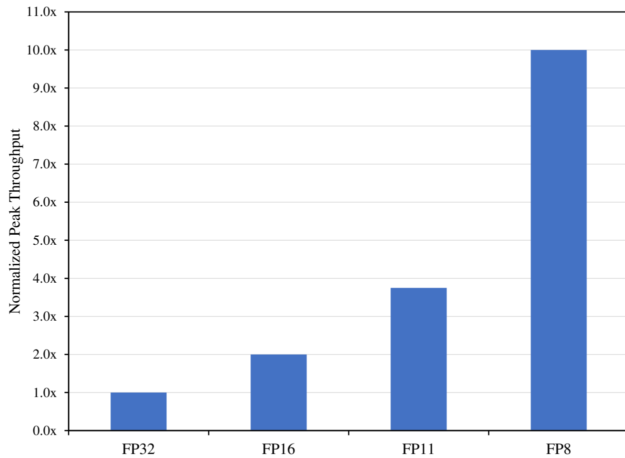

Figure 2: Normalized peak chip throughput vs. floating-point precision on Intel® Stratix® 10 FPGA

Fixed-point representation has been employed by NVIDIA [8], Xilinx [12], and Google's TPU [18] for CNN and RNN acceleration, while Microsoft has recently announced the use of a reduced-precision floating point on Intel® Stratix® 10 FPGAs in their acceleration of GRUs [7]. In Figure 2, we measure the impact of reduced-precision floating-point on overall compute performance available for a given Intel® Stratix® 10 FPGA. With reduced precision, fewer FPGA resources are required per compute unit.

While there have been many proposed solutions for achieving good accuracy at low precisions, there is no one-precision-fits-all solution. The ability to trade off precision for performance is heavily application- and topology-dependent. Every application has its own accuracy requirements, and thus values performance-accuracy tradeoffs differently. The different topologies used in these applications have different tolerance levels to reduced data precision. For example, SqueezeNet was designed to achieve AlexNet-level accuracies while reducing the model-size by 50× [14]; it is unsurprising that it is more sensitive to accuracy losses at reduced precision compared to AlexNet (see Figure 5). With its fine-grained, bit-level customizability, the FPGA platform can provide flexibility to employ a tuned precision-accuracy tradeoff that suits individual applications and topologies.

The DLA architecture leverages the FPGA platform to provide a reduced precision data representation with a tunable mantissa width (Figure 3). In addition to the ability to reduce the data width, the DLA architecture uses the FPGA fabric's bit-level granularity to improve resource usage efficiency by organizing floating-point numbers in "blocks" [2], defined as a group of FP numbers sharing a single common exponent. The "block size" is a tunable parameter that defines the number of mantissas sharing the common exponent. Placing a number into a block requires bit-shifting the mantissa such that it can use the block's common exponent, which may result in some precision loss. As can be seen in Figure 4, this loss of precision comes with the benefit of better resource utilization, as the storage requirements shrink when more numbers are placed in a block. The block size can be tuned to trade off accuracy for performance.

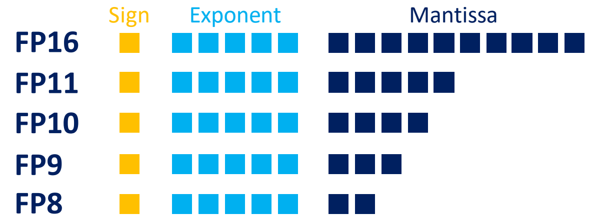

Figure 3: Reduced-precision floating point sign/exponent/mantissa breakdown.

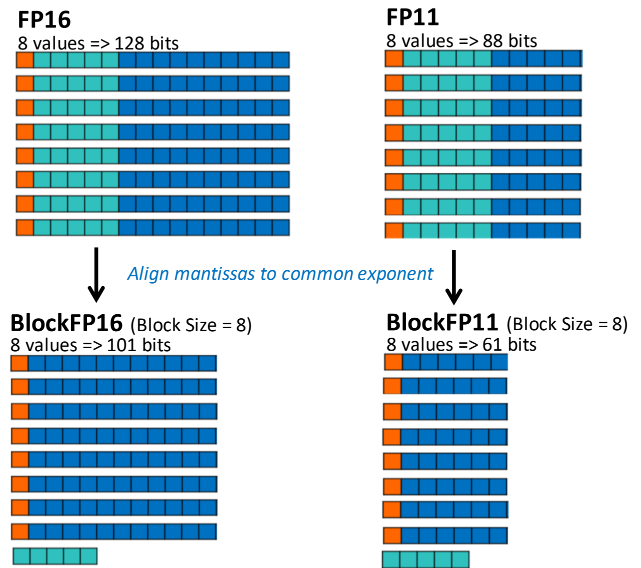

Figure 4: Organizing floating-point numbers in "blocks".

To investigate their accuracy impact, different precision and block sizes were tested on AlexNet, GoogleNet, SqueezeNet, ResNet-18 and VGG-16 [16]. As can be seen in Figure 5 and 6, each topology can tolerate different levels of precision loss. This illustrates how different topologies can benefit from reduced precision and bigger block sizes to varying degrees. By leveraging the flexibility of the FPGA, DLA can be tuned to the level of precision that best suits an application.

At a specific precision, there remains the challenge of implementing a PE architecture that maximizes throughput per area for that precision. The optimal PE for each precision-level is dependent on the FPGA platform; DSP and Logic Element (LE) architecture strongly influence how a PE can be designed to achieve the highest throughput at the lowest area cost. Choosing the right combination of DSPs and LEs for a certain precision is a complex problem that can be facilitated by CAD tools, as will be described in Sections 4 and 5.1. Scaling the design, once a precision and PE architecture is chosen, is done through compute parallelism, which is explored in the next section.

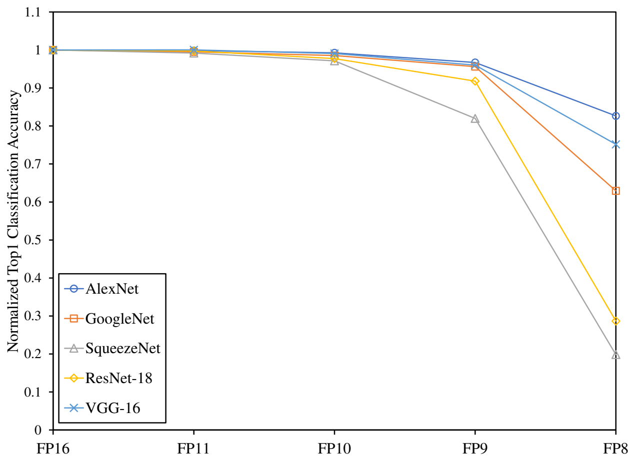

Figure 5: NN graphs have different tolerance levels to reduced precision. Accuracy numbers reported here are measured using a block size of 8.

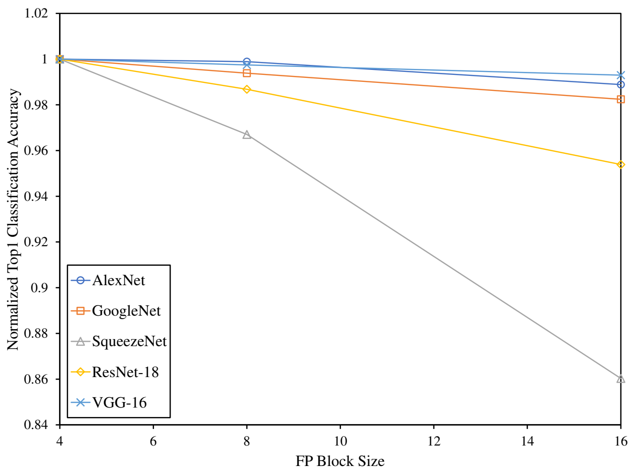

Figure 6: Increasing block size improves resource usage efficiency, but may cost accuracy depending on the graph used. Accuracy numbers reported here are measured using BlockFP9 precision.

3.2 Accelerator Geometry & Customization

To leverage the flexibility of FPGAs, we can customize the accelerator *vectorization* (degree of parallelism) to suit different classes of neural networks. A change to the vectorization manifests in a different accelerator *geometry* – the number of processing elements and the widths of databuses between them. Figure 1 shows the degrees of parallelism available in the accelerator, configurable via vectorization.

The problem size at each convolution stage is defined by the neural network topology:

- W, H, C are the input tensor width, height and depth.
- S, R, C are the filter width, height and depth.

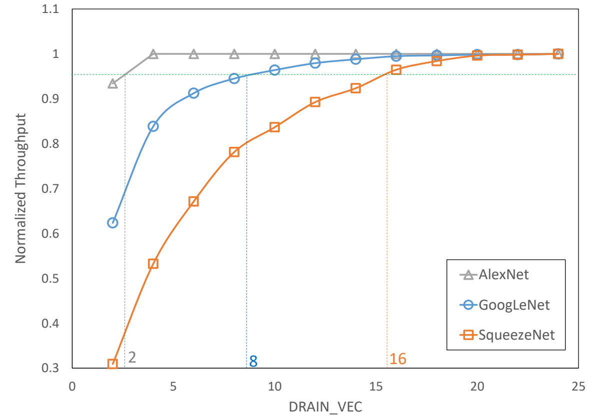

Figure 7: Effect of drain network vectorization on the throughput of GoogLeNet and SqueezeNet NNs, normalized to maximum performance.

- Q, P, K are the output tensor width, height and depth.

Architectural parameters in the accelerator control the degree of parallelism employed in the core compute, which impact both the number of compute units used as well as the width of datapaths:

- Q_VEC, P_VEC, and K_VEC define the window of the output tensor computed in parallel.
- C_VEC defines the depth of the input tensor read and used in parallel.
- S_VEC and R_VEC are the filter width and height read and computed in parallel.
- DRAIN_VEC is the width of the drain network from the PE array. This determines the subset of K_VEC that can be extracted from the PE array in one clock cycle.
- Each auxiliary kernel has an AUX_VEC, which describes the subset of the DRAIN_VEC that is processed in one clock cycle in each of the auxiliary kernels.

For each of our marquee neural networks, we customize the architecture to maximize throughput-per-area. We then analyze the geometry considerations for architectures tuned for various different graphs.

3.2.1 Smaller Filters Require Faster Drains. Each of the PEs consists of Q_VEC×P_VEC dot-product operations (each of which multiplies and sums C_VEC data items together), followed by accumulators. The accumulators keep accumulating the result of the dot products until it has iterated over the entire filter tensor; after which, a single output pixel is produced from a PE. Therefore, depending on the filter size and PE-array vectorization, the speed of producing results from the PEs differ – it depends on the ratio between the filter size and the vectorization of each PE.

Between the GoogLeNet and SqueezeNet networks, SqueezeNet has, on average, smaller filters. This causes convolution results to be produced more quickly from the PE array, and therefore requires a higher-bandwidth drain network. Figure 7 shows the effect of increasing the drain network parallelism (DRAIN_VEC) on the AlexNet, GoogLeNet and SqueezeNet networks. While GoogLeNet throughput saturates at DRAIN_VEC of 8, SqueezeNet requires DRAIN_VEC of 16 for 95% of maximum throughput. AlexNet only

needs a DRAIN_VEC of 2 to achieve 95% performance since its filters are the largest. Scaling DRAIN_VEC beyond these values yields diminishing returns as the plot shows. Supporting increased drain network parallelization also requires scaling up the area of the auxiliary kernels. Table 1 presents the area cost of scaling DRAIN_VEC. With a fixed function accelerator, a designer has to choose apriori between an additional 21% area penalty (across all networks) or a 70% performance penalty on SqueezeNet-like networks. With a reconfigurable device like an FPGA, the designer can only instantiate the larger DRAIN_VEC when required by the network being accelerated.

Table 1: Area cost of increasing DRAIN_VEC.

DRAIN_VEC	Aux Kernels Area (% of Full System)
2	4%
8	14%
16	25%

3.2.2 Interconnect Customization and Width Adapters Allow Building Only What is Needed. Another degree of flexibility exists in the auxiliary kernels domain – the flexible interconnect allows building only exactly what is needed for each graph. For example, the SqueezeNet Graph has no Local Response Normalization (LRN) layers, so we can remove that kernel completely. The interconnection pattern within the interconnect is also customizable based on the order of the auxilliary operations. For example, the AlexNet graph has both MaxPool and LRN layers, but LRN always comes first; whereas the GoogLeNet graph has some layers in which MaxPool precedes LRN, so we need to support both these connection patterns by adding more muxing and arbitration logic. The width of each of the auxiliary kernels can be customized separately based on how much bandwidth is required of each operation. Finally, DLA can also leverage FPGAs enhanced with hardened interconnects – such as embedded Networks-on-Chip [1, 3] – for high-bandwidth inter-kernel communication.

3.2.3 Balance Vectorization to Minimize Quantization Inefficiencies. In general, scaling up the tensor vectorization increases throughput at the expense of more area. Initially, the design was scaled by increasing K_VEC – this was relatively simple, since increasing K_VEC only entails adding more PEs. However, this method of scaling saw diminishing returns, as quantization inefficiencies can become more pronounced as vectorization dimensions increase. For example, if the output depth (K) of a layer is 96, and K_VEC is 64, this will require 2 complete iterations, and so the output depth will be snapped up to 128, with only 96/128 (75%) useful computations. On the other hand, if K_VEC is 32, the output depth divides perfectly into 3 iterations at 100% efficiency. To mitigate this quantization effect, it is possible to balance the scaling of the design across multiple different dimensions besides just K_VEC (e.g. P_VEC, Q_VEC, C_VEC, etc). The optimal balance of vectorization depends on the graph's layer dimensions. Figure 8 demonstrates this point by comparing throughput at two architectures of similar area for multiple different graphs. As can be seen in the figure, the optimal balance of scaling the design between P_VEC and K_VEC

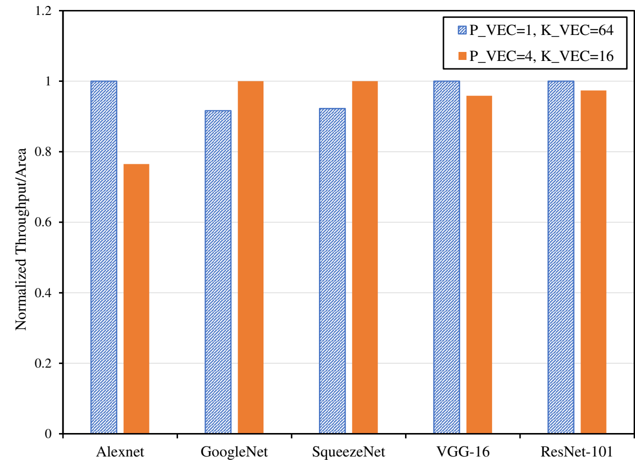

Figure 8: Throughput/Area on two architectures with different P_VEC and K_VEC vectorization.

varies based on the neural network topology being used. With a flexible FPGA platform, a custom instance of a deep learning accelerator can be can be generated, with vectorization parameters tuned for each application.

3.3 Memory Architecture

With their different models and layer sizes, neural networks have a wide range of memory requirements, and creating an efficient design around these requirements poses a complex problem. To support such requirements, the DLA architecture uses both the on-chip and off-chip memory provided by the FPGA platform.

On-chip memory provides a fast means to store and access data, avoiding the relatively long latency of communicating with off-chip memory. DLA uses on-chip memory for both filter and tensor data (Figure 1). Filter data is stored in a "filter cache" (FC) contained in each PE. While the PEs compute data, filters are pre-loaded from external memory into the filter caches for the next set of PE computation. The "stream buffer" is used to store intermediate tensors between layers on-chip. If the stream buffer is too small to hold a given tensor, then DLA falls back to using external memory. Given a limited DDR bandwidth, excessive use of external memory can bottleneck the performance of the design.

The balance between using on-chip and off-chip memory creates an important design tradeoff. Using more on-chip memory for the stream buffer alleviates DDR bottlenecks but consumes more chip resources that could have been used for more PEs. On the other hand, reducing on-chip memory used in the stream buffer, in favor of more PEs, provides more compute power at the cost of heavily relying on off-chip memory. The optimal balance of resources between on-chip memory and processing elements that maximizes performance is dependent on the graph's compute and memory requirements. Finding this optimal point requires accurate design modelling as well as knowledge of the target neural network and platform.

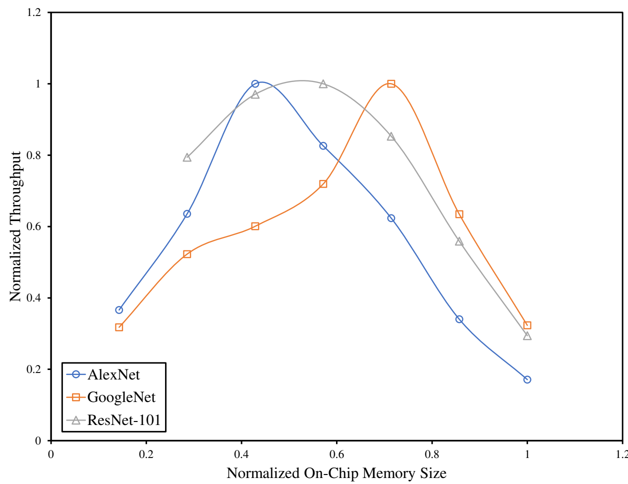

Figure 9: Impact of stream buffer memory vs. compute trade-off on AlexNet, GoogleNet and ResNet-101.

To illustrate this trade-off, we modelled the performance of AlexNet, GoogleNet and ResNet-101 at different stream buffer memory and PE sizes for a given FPGA platform. As can be seen in Figure 9, the optimal allocation of resources to memory and compute differs across each of the different graphs. This highlights the benefit of having a flexible FPGA fabric capable of specifically tailoring a design to achieve the optimal performance for a given application. A tradeoff that is optimal for one neural network can cause 40% or more performance degradation for a second neural network.

3.4 Implications of Flexibility

In general, one size does not fit all when designing deep learning accelerators. Using a flexible accelerator on a flexible FPGA platform can improve performance, with architecture parameters (compute precision, accelerator geometry, and memory architecture) tuned for a specific neural network topology. We have presented several dimensions of flexibility (by no means an exhaustive list); customization can deliver a performance benefit (or area savings) on each dimension. In aggregate, the benefit across the fully customized FPGA accelerator can outweigh the cost of the programmable fabric.

Flexibility of the deep learning acceleration architecture must be matched with flexible design-entry flow and CAD tools, to allow designers to specify optimized deep learning accelerators quickly.

4 HIGH- AND HIGHER-LEVEL DESIGN

With the large design space of possible architecture parameterizations, it becomes infeasible for a designer to manually configure the Deep Learning Accelerator using traditional FPGA RTL design techniques. We propose an extra level of abstraction to enable automatic specification and generation of custom deep learning acceleration. In Section 5, we discuss some of the implications on Physical Design when using such high-level design entry.

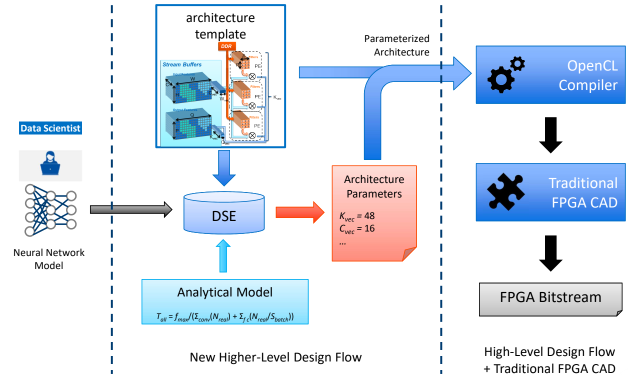

Figure 10: Design flow for neural network accelerators.

4.1 Abstracting Design Entry with OpenCL™

To achieve the desired flexibility, our Deep Learning Accelerator architecture is described in a generic form that can be easily modified with a set of architectural parameters. We describe our architecture using a set of pre-defined kernels written in OpenCL™, that can be modified with specific architectural parameters such as vectorization parameters, stream buffer dimensions, memory port widths, number of processing elements, and configuration of auxiliary kernels. The combination of these kernels form a generic architecture template. Once configured, the resulting architecture is compiled and executed through the Intel® FPGA SDK for OpenCL™. This design flow is illustrated on the right side of Figure 10.

High-level design tools like the Intel® FPGA SDK for OpenCL™ compile untimed, unscheduled OpenCL language into optimized RTL implementations, applying optimizations (pipelining, vectorization, compute unrolling) as necessary according to predicted system performance and knowledge of the physical FPGA architecture. [11] This abstraction is invaluable when creating complex, parameterizable, designs such as the Deep Learning Accelerator as it alleviates the need for time-consuming detailed physical design optimization and verification across a wide parameterization space.

4.2 Future Work: Even Higher-Level Design

Once we have an architecture template that can accept a wide range of architectural parameters to modify its structure, we envision that future users of the Deep Learning Accelerator will rely on even higher-level design techniques to select and configure appropriate architectures. An additional design space exploration (DSE) step is envisioned, to find the optimal architecture to run a given neural network model. The DSE step can be automated using traditional objective-optimization CAD techniques [15]. This would require a high-fidelity model to represent the physical characteristics of a given architecture and the resulting performance. Using the model, the DSE flow would efficiently search the architecture parameter space to identify the most optimal configuration for a given neural network model.

Each design point (a unique instance of the Deep Learning Accelerator) can be accurately modeled because our architecture is deterministic in nature, where all memory accesses, data movements, and compute times can be calculated apriori. The deterministic

architecture and analytical model is presented in [2], where both the resource utilization of the FPGA and the final performance of the design can be calculated prior to runtime.

An illustration of this proposed future approach is shown on the left side of Figure 10. Here, the DSE tool is provided the neural network model, architecture template, analytical model architecture, and objective function (such as performance or latency). The DSE tool would compile the neural network into a graph and map it to the architecture template core primitives. After sweeping through a wide range of architecture parameters, and generating performance estimates from the model architecture, a final set of parameters is chosen to satisfy the objective function. The chosen architecture parameters can be used to configure the architecture template, and the resulting design can be compiled through the OpenCL and traditional FPGA compilers to generate a bitstream for the FPGA.

This higher-level design flow, where neural network models are compiled to generate custom FPGA-based accelerators, has many implications on FPGA CAD, and introduces several new challenges that need to be resolved to ensure high-performance while enabling flexibility.

5 PHYSICAL DESIGN IMPLICATIONS

Today, the state of the art FPGA CAD tools provide automated placement, routing and packing of homogeneous resources– the three foundational CAD algorithms that account for the physical aspects of the FPGA fabric. Hierarchical physical design techniques, such as floorplanning, packing of heterogeneous resources, and partitioning are currently done manually by the designer. We study the manual application of these hierarchical techniques to our DLA architecture, and show the benefit. Consequently, we advocate for automation of these techniques, in future work on FPGA CAD (both high-Level design and traditional). This automation is required to support automatic exploration of large accelerator design spaces exposed by our higher-level design tools.

Our study shows the following benefits:

- Significantly improved Quality of Results (QoR), with improved scalability to larger circuits, higher performance, and lower resource utilization.
- Improved repeatability of compilation results, through the reduction of seed noise.
- Improved designer productivity on a large design via reuse of partitions and incremental compilations.

In the following sections, we describe the techniques that we have manually prototyped in our physical design study that we applied to DLA.

5.1 Packing of Heterogeneous Resources for Efficient Implementation of PEs

The PEs occupies the majority of the area of the FPGA. The first challenge is to achieve high utilization of heterogeneous resources within that area, i.e., utilize nearly all available Logic Elements (LEs), DSP Blocks (DSPs) and block memory (M20Ks) in that area. To achieve this, each PE (or a small number of PEs) should consume a mix of LE, DSP and M20K resources that matches the resources within the physical region where they are implemented. Alternatively, this challenge can be formulated as a packing problem: given a mix of heterogeneous resource in a small region of a device, maximize the number of PEs that can be packed into that region.

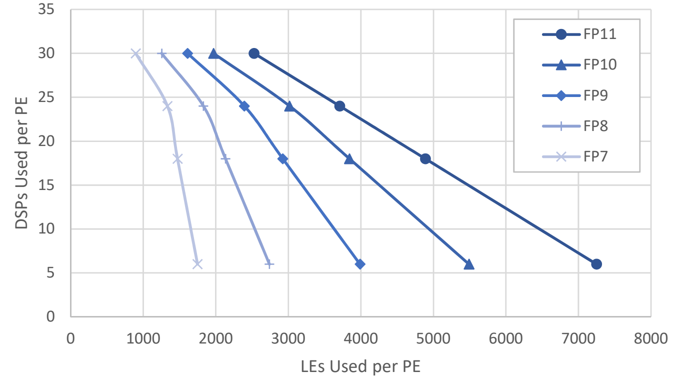

Figure 11: LE vs. DSP Resource Trade-off Curves for Equivalent Implementations of PEs, Across Different Precisions

One approach employed in order to address this challenge utilizes multiple variants of dot products, which are the building blocks of PEs. Each dot product variant uses a different tradeoff of LE and DSP resources. Figure 11 shows the tradeoff curves available when looking at multiple variants of equivalent dot products at different precisions. We can then solve an integer linear programming problem to select a mix of dot product variants that maximize the available resources given a region constraint. Today, the design of the different dot product variants is done manually at specific architecture design points, as well as the formulation of the linear system.

In order to enable the higher-level abstractions described in Section 4, next-generation tools should strive to automate this process– from the user's description of the dot product mathematics, the tool should generate and optimize a mix of dot product variants that maximizes resource usage, transparently to the user.

5.2 Systolic Arrays for Scalable Circuits

The use of systolic arrays and mesh-like strcuctures on FPGAs is motivated by the need to exploit the massive parallelism of modern FPGAs in a scalable manner [5, 17, 20]. Systolic arrays are circuits with regular structure and nearest-neighbor connections, primarily used in algorithms where the same piece of data needs to be broadcast and reused in a large number of PEs. In our DLA systolic array design, the broadcast structure is designed as a linear data forwarding pipeline in which data flows from one PE to another. This avoids a large fan-out of a wide data-path from a central location, and efficiently leverages local routing without introducing routing congestion.

In current tools, the user manually expresses the systolic array structure explicitly in their code by designing the data forwarding mechanism between the neighboring PEs. Future work could explore techniques for inferring large broadcast structures in user code, and automatically converting these structures to systolic arrays, to support the large design spaces of accelerators.

5.3 Floorplanning for Improving Performance of Large Circuits

Although the systolic array of PEs are a regular and repeated structure, we observe inefficiencies when compiling large systolic arrays which fill entire FPGA devices. The analysis of the placement of PEs by the CAD tool shows that in some cases the neighboring PEs are not placed close to each other. This results in long routes between PEs and a drop in frequency and performance.

We can address this challenge by manually creating floorplans for the systolic array, which are feasible due to its regular structure. Our floorplans comprise of a number of physical regions, we can then assign PEs to physical regions such that any two neighboring PEs are either in the same region or in two adjacent regions. This results in localized placement of PEs and mitigates the drop in performance for larger systolic arrays. We measure, on average, a 30% difference in performance for unfloorplanned designs versus a design floorplanned in this fashion. A similar benefit of floorplanning has been shown in a earlier work on floorplanning of mesh-of-functional-units overlay floorplanning [6].

Although manual floorplanning is feasible for a single architecture parameterization on a single FPGA device, it is not feasible across the full architecture design space and target devices. Automating this class of manual floorplanning effort remains a key outstanding physical design challenges for us, and research efforts that show promise have already started in academia [19].

5.4 Designer Productivity: Partitioning, Reuse and Incremental Compilation

As the sizes of FPGAs grow exponentially, the size of circuits implemented on the FPGA grow at the same pace. In addition to the QoR scaling challenges, these large sizes lead to long compile times, which can be up to a day. This poses two significant challenges for designer productivity. The first is development cycle, as the designer can only do a single iteration per day. The second is a reduced repeatability of results, as even a small design change can result in performance changes, due to seed noise inherently present in the CAD tool.

To enable scale, we employ approaches that leverage design partitioning. The DLA is composed of multiple OpenCL kernels, each encapsulating a distinct functionality of the design. The use of design partitioning can improve designer productivity by both reducing compile time and improving repeatability. By preserving the post-place-and-route netlists of partitions that have not been modified, only the kernels that the designer modified need to be re-compiled. Since in most cases only a small part of the design needs to be re-compiled, it is more likely that a single-seed compile will close timing.

When considering the large DLA architecture design space, the ideal partitioning may not be possible to define apriori, due to changing kernel size and number across all the architecture variants. Further research on automating the process of dynamic design partitioning (for an arbitrary set of interconnected kernels of varying sizes) is a key direction for future work. In addition to improving the development speed and repeatability of QoR, it would relieve designers from the burden of manual design partitioning.

6 CONCLUSION

We have shown through a case study of our Deep Learning Accelerator that flexibility to customize an architecture is essential to create future-proof accelerators. The resulting design space requires higher-level abstractions for efficient design and exploration. We advocate for new FPGA design flows that generate custom architectures from a high-level neural network description, and improved physical design flows to infer systolic arrays, optimize for resource constraints, and automate floorplanning and partitioning. With these future enhancements, FPGAs can continue to scale, and be an accelerator of choice for applications such as deep learning.

REFERENCES

[1] Mohamed S Abdelfattah, Andrew Bitar, and Vaughn Betz. 2015. Take the highway: Design for embedded NoCs on FPGAs. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 98–107.

[2] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C. Ling, and Gordon R. Chiu. 2017. An OpenCL™Deep Learning Accelerator on Arria 10. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17)*. ACM, New York, NY, USA, 55–64.

[3] Andrew Bitar, Mohamed S Abdelfattah, and Vaughn Betz. 2015. Bringing programmability to the data plane: Packet processing with a NoC-enhanced FPGA. In *Field Programmable Technology (FPT), 2015 International Conference on*. IEEE, 24–31.

[4] Diane M. Bryant. 2016. Keynote at Intel Developer's Forum 2016, San Francisco. (August 2016). https://newsroom.intel.com/chip-shots/2016-idf-keynotes-innovation-drives-technology-future-artificial-intelligence/

[5] D. Capalija and T. S. Abdelrahman. 2013. A high-performance overlay architecture for pipelined execution of data flow graphs. In *2013 23rd International Conference on Field programmable Logic and Applications*. 1–8.

[6] D. Capalija and T. S. Abdelrahman. 2014. Tile-based bottom-up compilation of custom mesh-of-functional-units FPGA overlays. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. 1–8.

[7] Eric et. al Chung. 2017. Accelerating persistent neural networks at datacenter scale. HotChips.

[8] NVidia Corporation. 2017. NVidia TensorRT. (2017).

[9] C. Szegedy et al. 2015. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1–9.

[10] Olga Russakovsky et al. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252.

[11] T. S. Czajkowski et al. 2012. From opencl to high-performance hardware on FPGAS. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*. 531–534.

[12] Yao et. al Fu. 2016. Deep Learning with INT8 Optimization on Xilinx Devices. *white paper of Xilinx* (2016).

[13] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[14] Forrest N et al. Iandola. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[15] Jacopo Panerati, Donatella Sciuto, and Giovanni Beltrame. 2017. *Handbook of Hardware/Software Codesign: Optimization Strategies in Design Space Exploration*. Springer, Netherlands.

[16] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[17] Xuechao et al. Wei. 2017. Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs. In *Proceedings of the 54th Annual Design Automation Conference 2017 (DAC '17)*. ACM, New York, NY, USA, Article 29, 6 pages.

[18] Yonghui et al. Wu. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

[19] Xiaodong Xu, Qi Xu, Jinglei Huang, and Song Chen. 2017. An Integrated Optimization Framework for Partitioning, Scheduling and Floorplanning on Partially Dynamically Reconfigurable FPGAs. In *Proceedings of the on Great Lakes Symposium on VLSI 2017 (GLSVLSI '17)*. ACM, New York, NY, USA, 403–406.

[20] Jialiang Zhang and Jing Li. 2017. 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 (FPGA '17)*. ACM, New York, NY, USA, 25–34.