WP504 (v1.0.1) October 14, 2018

Accelerating DNNs with Xilinx Alveo Accelerator Cards

The Xilinx xDNN processing engine, using Xilinx Alveo Data Center accelerator cards, is a high-performance energy-efficient DNN accelerator and outperforms many common CPU and GPU platforms today in raw performance and power efficiency for real-time inference workloads. The xDNN processing engine is delivered through the ML Suite available on many cloud environments, such as AWS EC2 or Nimbix NX5.

ABSTRACT

The Xilinx® Deep Neural Network (xDNN) engine provides high-performance, low-latency, energy-efficient DNN acceleration using Xilinx® Alveo™ Data Center accelerator cards. By keeping energy costs low and minimizing the number of specific accelerators needed in the implementation, total cost of ownership (TCO) can be significantly reduced.

Xilinx Alveo accelerator cards excel at high-performance, energy-efficient, flexible Machine Learning (ML) inference. The xDNN processing engine has been developed to generically execute Convolutional Neural Networks (CNNs) like ResNet50, GoogLeNet v1, Inception v4—even CNNs containing custom layers.

This white paper presents an overview of the xDNN hardware architecture and software stack, as well as benchmarking data that supports the claim of "Best in Class" energy-efficient inference.

Guidance to the reader is provided to enable re-creation of the results on the Alveo Data Center Accelerator Card.

Deep Learning Relevance in Data Center Applications

Deep learning methodologies have found tremendous success in various application domains over the past few years. One area of applied Machine Learning (ML) is vision and video processing. Video content on the internet has also grown rapidly over the past few years, and the need for methods of sorting, classifying, and identifying imagery has grown commensurately.

The Convolutional Neural Network (CNN), a type of ML neural network, has been an effective method for processing image data, especially in the context of Data Center deployment. CNN image networks can be used to classify and analyze images in the cloud. In many cases, the latency of image processing is critical to the end application, such as flagging illegal content in streaming video.

This white paper describes the Xilinx Deep Neural Network (xDNN) engine, a programmable inference processor capable of low-latency, energy-efficient inference running on Xilinx Alveo accelerator cards. The xDNN inference processor is a generic CNN engine that supports a wide variety of standard CNN networks. The xDNN engine integrates into popular ML frameworks such as Caffe, MxNet, and TensorFlow through the Xilinx xfDNN software stack. The xDNN processing engine running on an Alveo accelerator card is capable of 4,000 or more images per second of GoogLeNet v1 throughput, which translates to over 70% computational efficiency at Batch = 1.

As this computational efficiency suggests, xDNN running on Xilinx Alveo accelerator cards can outperform acceleration platforms such as GPUs for low-latency inference. It is widely known that GPU platforms can increase their performance by batch processing many images together; however, while batch processing improves performance and reduces required GPU memory bandwidth, the side effect of batching is a significant increase in latency.

In comparison, the xDNN processing engine does not rely on batching to achieve maximum throughput performance. Each engine runs independently, and weight memory is unshared. Each engine operates at Batch = 1, and several engines can be implemented on a single Alveo Accelerator Card. Thus, increasing the number of xDNN engines in the device only increases aggregate device Batch=1 throughput.

xDNN Architecture Overview

The xDNN hardware architecture is shown in Figure 1. Each xDNN engine is made up of a systolic array, instruction memory, execution controller, and element-wise processing units. The engine receives tensor instructions from command software executing on the host processor through the instruction queue. The instructions (tensor and memory operations) for the CNN network change only if the target network changes. Repeated execution of the same network reuses the previously loaded instructions resident in the instruction buffer.

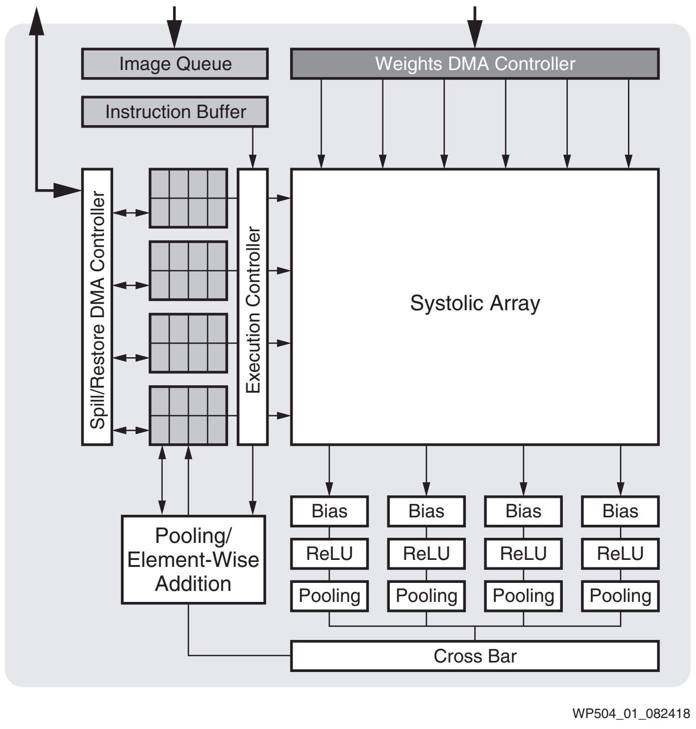


Figure 1: **xDNN Hardware Architecture**

xDNN Processing Engine Architectural Highlights

- Dual Mode: Throughput-Optimized or Latency-Optimized
- Command-Level Parallel Execution
- HW-Assisted Image Tiling
- Custom Layer Support (Heterogeneous Execution)
- Systolic Array Architecture

Throughput and Latency Optimized Modes

One of the architectural features of xDNN processing engine includes two operational modes, one for throughput optimization and another for latency optimization. In throughput optimized mode, data flow parallelism is exploited by creating an optimized processing engine (PE) to handle specific layers that map inefficiently to a general systolic array.

For example, the first layer of GoogLeNet v1 is an RGB layer, which represents nearly 10% of the overall compute overhead, does not map efficiently to a systolic array that efficiently computes the remainder of the network. In this throughput optimized mode, xDNNv3 includes an additional systolic array customized for three input channels. The net effect of this change is higher overall compute efficiency because the first layer of the next image can be computed while the previous image convolution and FC layers complete their respective processing.

For applications where the lowest single-image latency is desired, users have the option of deploying a latency-optimized version of the engine. For those applications, xDNN PE pipelining can be adjusted to reduce latency.

Command-Level Parallel Execution

The xDNN processing engine has dedicated execution paths for each type of command (download, conv, pooling, element-wise, and upload). This allows for convolution commands to be run in parallel with other commands if the network graph allows it. Certain network graphs have parallel branches of different instruction types that sometimes allow for parallel processing. For example, in the GoogLeNet v1 inception module, the 3x3 max pooling layer is a prime example of a layer that can be run in parallel with the other 1x1/3x3/5x5 convolutions using the xDNN processing engine. Figure 2 shows the inception module of the GoogLeNet v1 network.

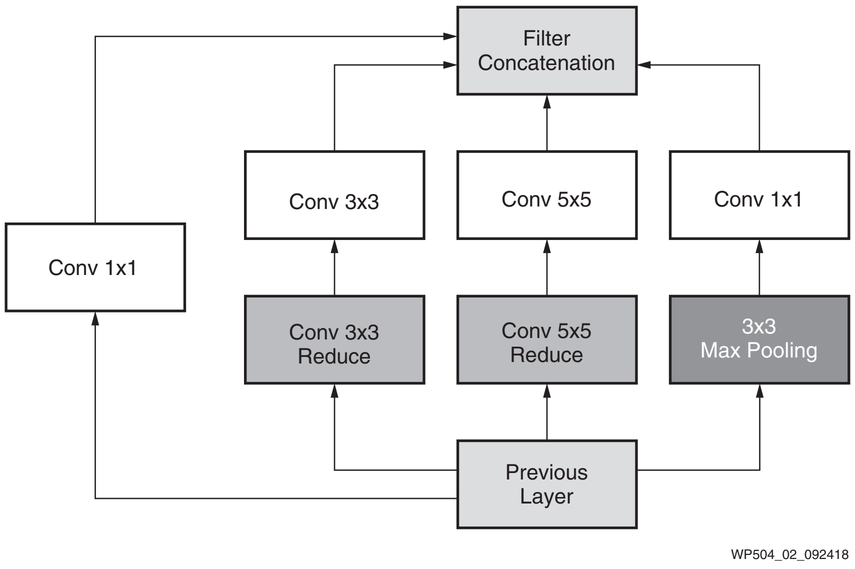


Figure 2: **Inception Layer in GoogLeNet v1**

As shown in Figure 3, the software can schedule 3x3 max pooling in parallel with 3x3 convolution of the second branch.

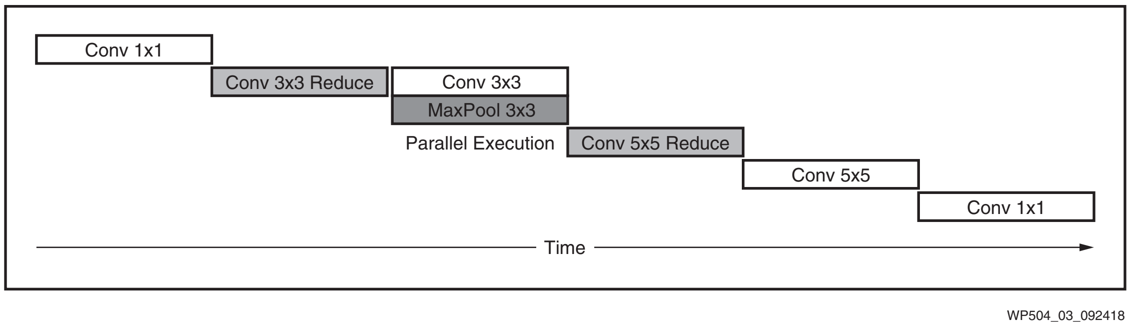


Figure 3: **xDNN Scheduling of Inception Layer in GoogLeNet v1**

Hardware-Assisted Image Tiling

The xDNN processing engine has a built-in hardware-assisted image tiling feature to support networks with large image/activation sizes. The xDNN processing engine allows input feature map tiling across both width and height. This is illustrated in Figure 4.

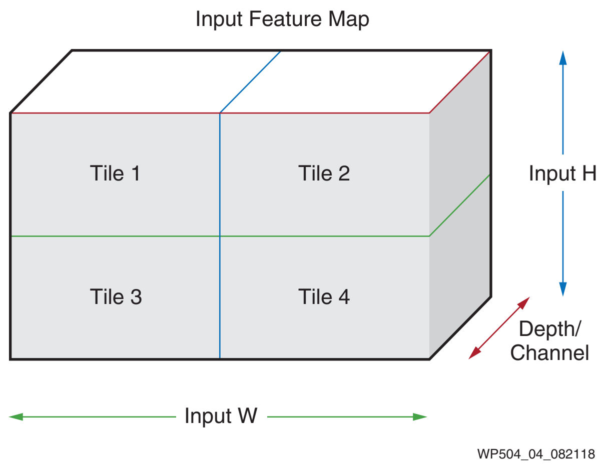


Figure 4: **Hardware-Assisted Image Tiling Feature**

Hardware-assisted image tiling takes a single non-data move instruction (Conv, Pool, EW) and generates the correct sequence of micro-operations (Download, Operation, Upload). The micro-operations are fully pipelined in hardware by logically partitioning activation memory into two regions, like a double buffer.

Custom Network Support through Heterogeneous Execution

Even though the xDNN processing engine supports a wide range of CNN operations, new custom networks are constantly being developed—and sometimes, select layers/instructions might not be supported by the engine in the FPGA. Layers of networks that are not supported in the xDNN processing engine are identified by the xfDNN compiler and can be executed on the CPU. These unsupported layers can be in any part of the network—beginning, middle, end, or in a branch.

Figure 5 shows how the processing can be partitioned by the compiler onto various PEs within the xDNN processing engine, or even the CPU.

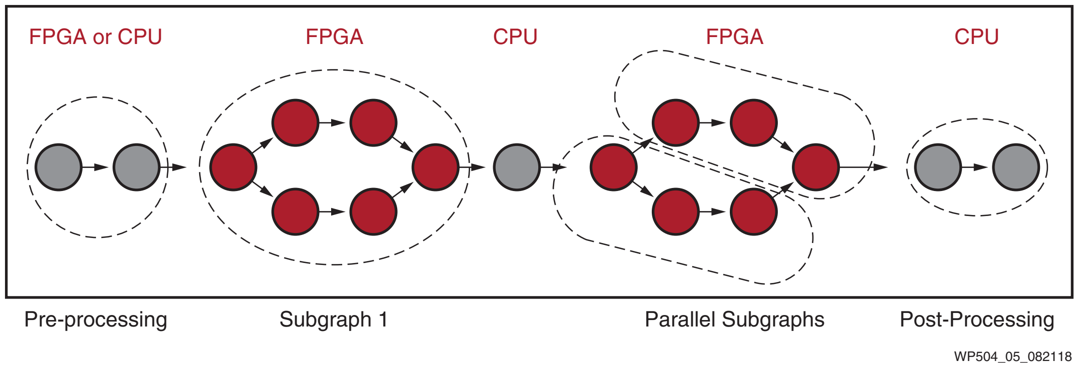


Figure 5: **Processing Partitioned by the Compiler**

Systolic Array Architecture

The xDNN processing engine leverages techniques, such as those described in the "SuperTile" paper[1], to achieve a high operating frequency. This SuperTile DSP macro provides a relationally placed macro that can be tiled to build larger compute arrays such as matrix multiplication and convolutions, the most compute-intensive operations of a CNN.

Figure 6 shows an example of the logical processing element being mapped into the DSP48 and CLB-M (LUTRAM) tiles in the FPGA. This macro cell is the fundamental processing unit within the xDNN systolic array.

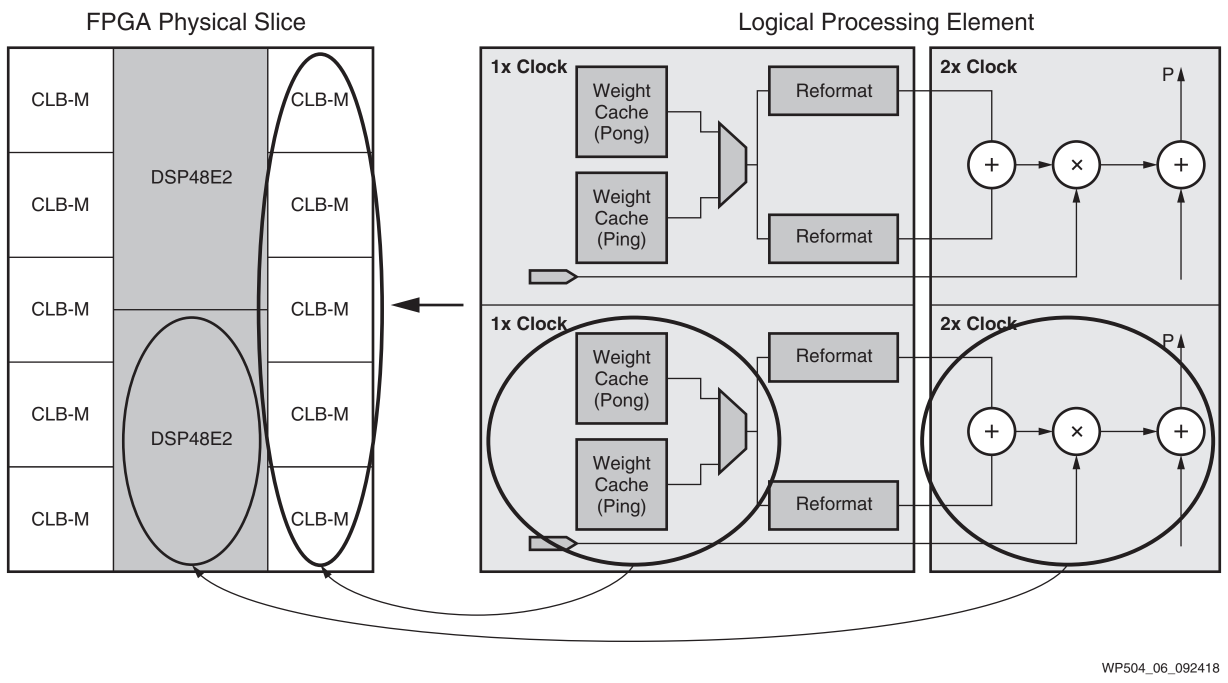


Figure 6: **MAC and Weight Packing in DSP Macro Example**

1. E. Wu et al., Xilinx Inc. IEEEXplore Digital Library, Sept. 2017, *A High-Throughput Reconfigurable Processing Array for Neural Networks.*

xfDNN Software Stack Overview

The xfDNN software stack is a combination of software tools and APIs that enable seamless integration and control of the xDNN processing engine with a variety of common ML frameworks. The flow diagram in Figure 7 details how networks and models are prepared for deployment on xDNN through Caffe, TensorFlow, or MxNet. The xfDNN compiler supports layers for xDNN while running unsupported layers on the CPU. After the network/models have been compiled and quantized—a process that typically takes under a minute—the user can interface with the xDNN processing engine through a selection of simple-to-use Python or C++ APIs.

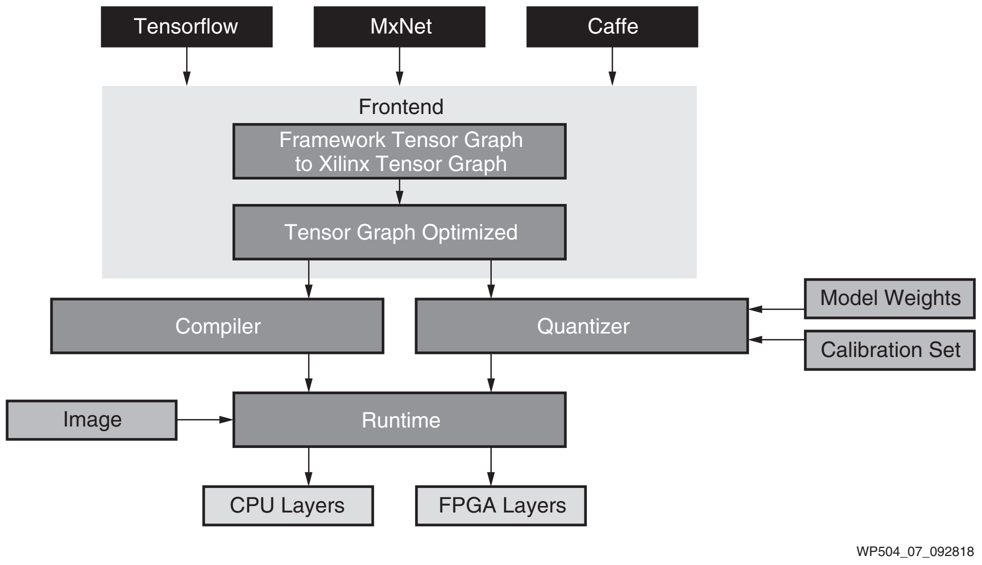


Figure 7: **xfDNN Flow Diagram**

The Xilinx xfDNN software stack comprises:

1. Network Compiler and Optimizer

The compiler produces an instruction sequence to be executed on the xDNN engine, which provides the tensor-level control and data flow management to implement the given network.

2. Model Quantizer

The quantizer produces a target quantization (INT8 or INT16) from a trained CNN network model without requiring hours of retraining or a labeled dataset.

3. Runtime and Scheduler

xfDNN simplifies communicating and programing the xDNN processing engine and utilizes an SDx-compliant runtime and platform.

Figure 8 shows a flow diagram of the xfDNN library, connecting deep-learning frameworks with xDNN IP running on a Xilinx FPGA.

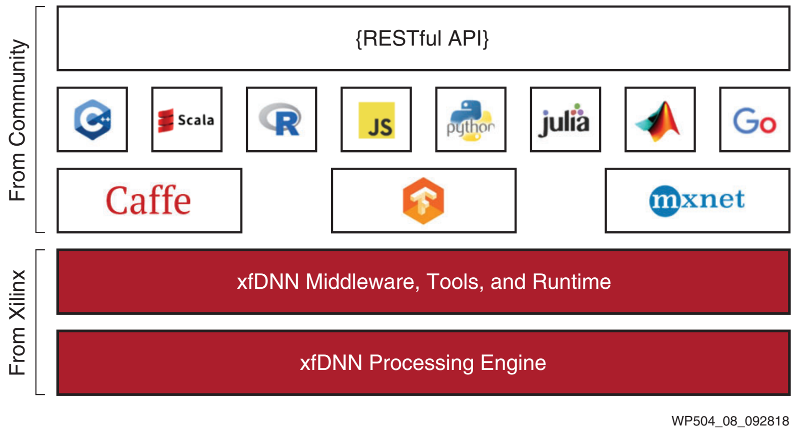


Figure 8: **xfDNN Software Stack**

More on the xfDNN Compiler

Modern CNNs are graphs of hundreds of individual operations—i.e., Convolution, Maxpool, Relu, Bias, Batch Norm, Elementwise Add, etc. The primary job of the compiler is to analyze CNN networks and produce an optimized set of instructions to execute on xDNN.

The xfDNN compiler not only provides simple Python APIs to connect to high-level ML frameworks, but it also provides tools for network optimization by fusing layers, optimizing memory dependencies in the network, and pre-scheduling the entire network. This removes CPU host control bottlenecks. See Figure 9 as an example of this.

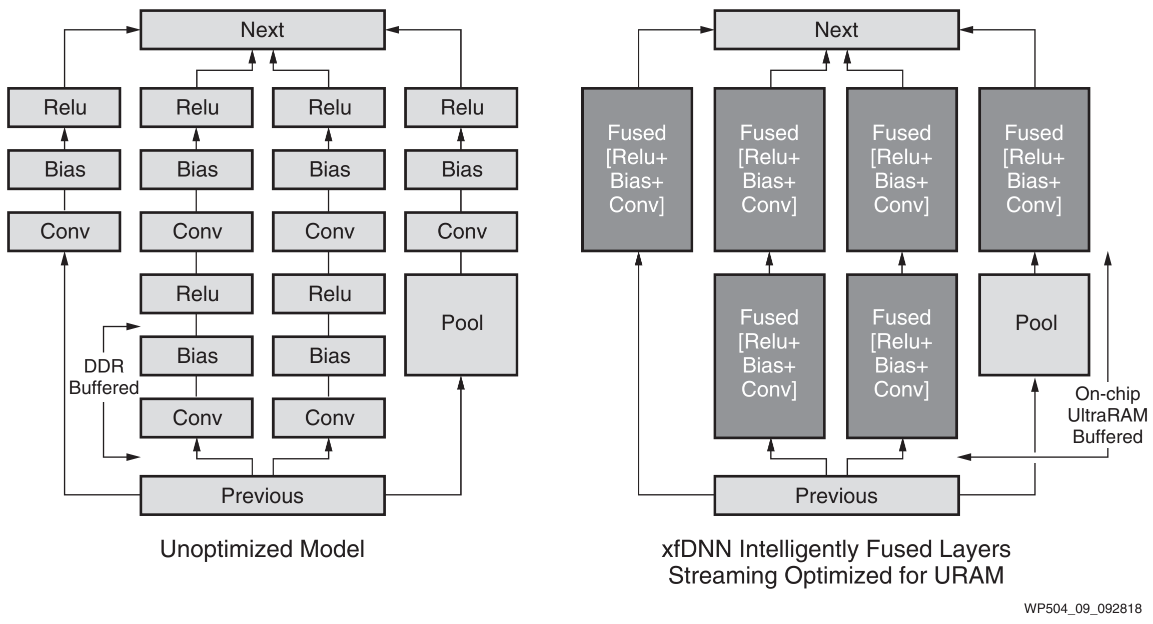
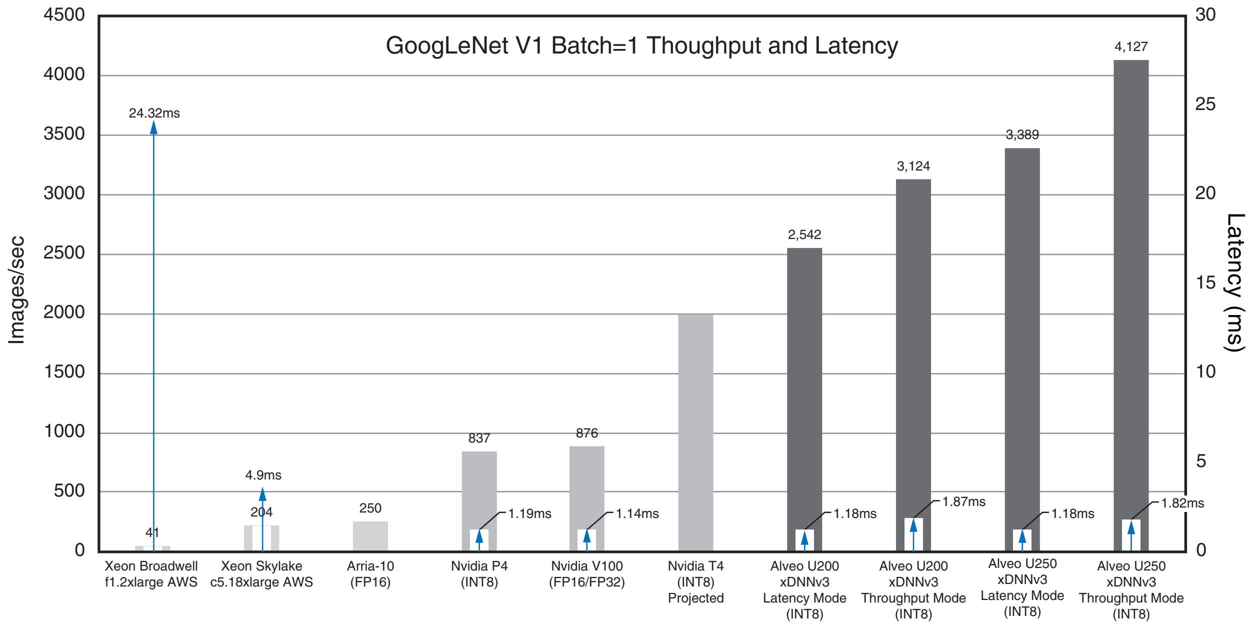


Figure 9: **xfDNN Compiler Optimizations**

Performance Benchmarking Results

With the growing number of real-time AI services, latency becomes an important aspect of overall AI service performance. Unlike GPUs, where there is a significant trade-off between latency and throughput, xDNNv3 DNN engines can provide both low-latency *and* high-throughput. In addition, xDNNv3 kernels provide simple Batch=1 interfacing, which reduces complexities in interfacing software by not requiring any queuing software to auto-batch input data to achieve maximum throughput.

Figure 10 and Figure 11 show CNN, latency, and throughput benchmarks on Alveo accelerator cards and popular GPU and FPGA platforms. Figure 10 shows GoogLeNet V1 Batch=1 throughput measured in images per second along the left Y-axis. The number shown above the throughput is measured/reported latency in milliseconds.



Notes:

1. Xeon E5-2696 v4 f1.2xlarge AWS instance, Ubuntu 16.04LTS, amd64 xenial image built on 2018-08-14, Intel Caffe (https://github.com/intel/caffe), Git Version: a3d5b02, run_benchmark.py w/ Batch=1 modification.
2. Xeon Platinum 8124 Skylake, c5.18xlarge AWS instance, Ubuntu 16.04LTS, amd64 xenial image built on 2018-08-14, Intel Caffe, Git Version: a3d5b02, run_benchmark.py w/ Batch=1 modification.
3. Arria-10 numbers taken Intel White Paper, "Accelerating Deep Learning with the OpenCL™ Platform and Intel Stratix 10 FPGAs." https://builders.intel.com/docs/aibuilders/accelerating-deep-learning-with-the-opencl-platform-and-intel-stratix-10-fpgas.pdf. Arria latency figures have not been published.
4. Nvidia P4 and V100 numbers taken from Nvidia Technical Overview, "Deep Learning Platform, Giant Leaps in Performance and Efficiency for AI Services, from the Data Center to the Network's Edge." https://images.nvidia.com/content/pdf/inference-technical-overview.pdf. Data retrieved on September 3, 2018.
5. Nvidia T4 projection based on current available published benchmark. GoogLeNet Batch=1 performance range between 1700-2000 images/sec based on early power efficiency benchmarks.
6. Alveo U200 numbers measured Intel Xeon CPU E5-2650v4 2.2GHz, 2400MHz DDR4, Ubuntu 16.04.2 LTS Instance running on OpenStack Pike, Centos 7.4, Pre-release Version of MLSuite, streaming_classify.py, synthetic data, MLSuite DSA Thin Shell, FC and SoftMax layers running on Xeon Host and operations not included in compute totals (0.06% of overall compute).
7. Alveo U250 numbers measured Intel Xeon Silver 4110 CPU @ 2.10GHz, CentOS Linux release 7.4.1708, Pre-release version of MLSuite, streaming_classify.py, synthetic data, DSA: ML Thin Shell, FC and SoftMax layers running on Xeon Host and Operations not included in compute totals (0.06% of overall compute).

WP504_10_092418

Figure 10: **GoogLeNet v1 Batch=1 Throughput**

Figure 11 shows GoogLeNet V1 throughput measured in images per second per watt along the Y axis.

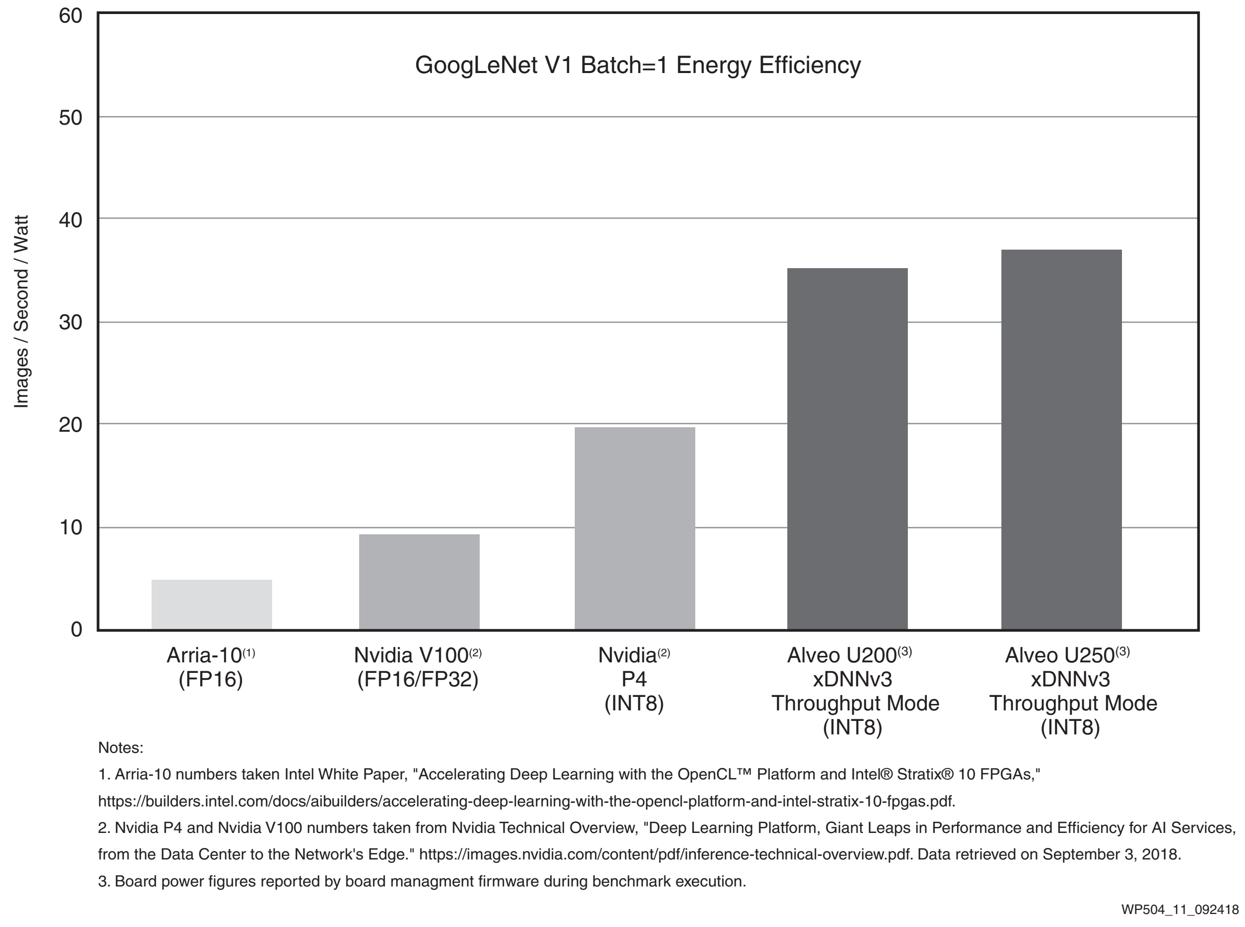


Figure 11: **GoogLeNet v1 Batch=1 Energy Efficiency**

While GoogLeNet v1 performance is shown for benchmarking purposes, xDNN supports a broad range of CNN networks. Refer to ML Suite documentation (https://github.com/Xilinx/ml-suite) for more information on running other CNN networks.

Conclusion and Call to Action

As shown in the shared performance results, the xDNN processing engine is a high-performance, energy-efficient DNN accelerator that outperforms many common CPU/GPU platforms today for real-time inference workloads. The xDNN processing engine is available through the ML Suite on many cloud environments, such as Amazon AWS/EC2 or Nimbix NX5. It scales seamlessly to on-premises deployment through Xilinx's new Alveo accelerator cards.

Xilinx's reconfigurable FPGA silicon allows users to continue receiving new improvements and features through xDNN updates. This allows the user to keep up with changing requirements and evolving networks.

For more information about getting started, go to: https://github.com/Xilinx/ml-suite or https://www.xilinx.com/applications/megatrends/machine-learning.html

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
10/14/2018	1.0.1	Editorial update.
10/02/2018	1.0	Initial Xilinx release.

Disclaimer

Automotive Applications Disclaimer