Accelerating Convolutional Neural Network With FFT on Embedded Hardware

Tahmid Abtahi, Colin Shea, Amey Kulkarni, and Tinoosh Mohsenin

***Abstract*— Fueled by ImageNet Large Scale Visual Recognition Challenge and Common Objects in Context competitions, the convolutional neural network (CNN) has become important in computer vision and natural language processing. However, state-of-the-art CNNs are computationally memory-intensive, thus energy-efficient implementation on the embedded platform is challenging. Recently, VGGNet and ResNet showed that deep neural networks with more convolution layers and a few fully connected layers can achieve lower error rates, thus reducing the complexity of convolution layers is of utmost importance. In this paper, we evaluate three variations of convolutions, including direct convolution (Direct-Conv), fast Fourier transform (FFT)-based convolution (FFT-Conv), and FFT overlap and add convolution (FFT-OVA-Conv) in terms of computation complexity and memory storage requirements for popular CNN networks in embedded hardware. We implemented these three techniques for ResNet-20 with the CIFAR-10 data set on a low-power domain-specific many-core architecture called power-efficient nanoclusters (PENCs), NVIDIA Jetson TX1 graphics processing unit (GPU), ARM Cortex A53 CPU, and SPARse Convolutional NETwork (SPARCNet) accelerator on Zynq 7020 FPGA to explore the tradeoff between software and hardware implementation, domain-specific logic and instructions, as well as various parallelism across different architectures. Results are evaluated and compared with respect to throughput per layer, energy consumption, and execution time for the three methods. SPARCNet deployed on Zynq FPGA achieved 42-ms runtime with 135-mJ energy consumption with a 10.8-MB/s throughput per layer using FFT-Conv for ResNet-20. Using built-in FFT instruction in PENC, the FFT-OVA-Conv performs 2.9× and 1.65× faster and achieves 6.8× and 2.5× higher throughput per watt than Direct-Conv and FFT-Conv. In ARM A53 CPU, FFT-OVA-Conv achieves 3.36× and 1.38× improvement in execution time and 2.72× and 1.32× higher throughput than Direct-Conv and FFT-Conv. In TX1 GPU, FFT-Conv is 1.9× faster, 2.2× more energy-efficient, and achieves 5.6× higher throughput per layer than Direct-Conv. PENC is 10 916× and 1.8× faster and 5053× and 4.3× more energy-efficient and achieves 7.5× and 1.2× higher throughput per layer than ARM A53 CPU and TX1 GPU, respectively.**

***Index Terms*— Convolutional neural network (CNN), deep learning, domain-specific many-core accelerator, energy-efficient, FFT overlap and add, field-programmable gate array (FPGA), graphics processing unit (GPU).**

Manuscript received September 10, 2017; revised January 14, 2018; accepted February 20, 2018. Date of publication June 21, 2018; date of current version August 23, 2018. This work was supported by the National Science Foundation under CAREER Award 1652703. *(Corresponding author: Tahmid Abtahi.)*

The authors are with the Department of Computer Science & Electrical Engineering, University of Maryland at Baltimore County, Baltimore, MD 21250 USA (e-mail: abtahi1@umbc.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2018.2825145

I. Introduction

DEEP neural networks have been shown to outperform prior state-of-the-art solutions that rely heavily on hand-engineered features coupled with simple classification techniques [1]. In addition to achieving improvement in accuracy, they offer a number of additional benefits, such as the ability to perform end-to-end learning by performing both hierarchical feature abstraction and inference [2]–[7].

Due to these advantages, they are now the reference architecture in computer vision [8], natural language processing [9], and speech recognition applications [10]. Over the past four years, an enormous amount of research has been conducted from two different perspectives:

1) algorithm improvement, to reduce error rate on ImageNet Large Scale Visual Recognition Challenge data set [11], [12], target-specific applications, such as object detection and tracking [13], face identification [14], localization, and mapping [15], and introducing layers to lower computational and time complexity of training and inference [16]–[18];
2) hardware improvement, to reduce memory transfers [19]–[21] and computations of convolution layer [4], [5], [22], using heterogeneous platforms with convolution accelerators, such as Synopsys designware EV6x, Movidius myriad2, and CEVA XM6.

Embedded applications, including driverless cars and drones, have different types of sensors onboard, including LiDARs, stereo, vision cameras, and radar, which generate a significant amount of data each minute [23]–[26]. At the same time, these applications require accurate decision-making abilities with additional rigid constraints on real time and power consumption and a very little space on board [27]. Deep convolutional neural networks (CNNs) are an ideal candidate for such applications; however, these networks experience real time memory and power consumption bottlenecks [28].

In CNNs, two layers mainly contributed to network bottlenecks, and the convolution layer is the most computationally complex, whereas the fully connected (FC) layer is the most memory-intensive. For example, Fig. 1 shows layerwise computation breakdown for AlexNet, a popular object detection network. Almost 90% of the computation is dominated by convolution operation which is computationally intensive. Therefore, energy-efficient and low-latency acceleration of the CNN is extremely important.

There has been a surge in research in deep CNN accelerators. Page *et al.* [22] presented SPARse Convolutional NETwork (SPARCNet), an FPGA-based CNN accelerator

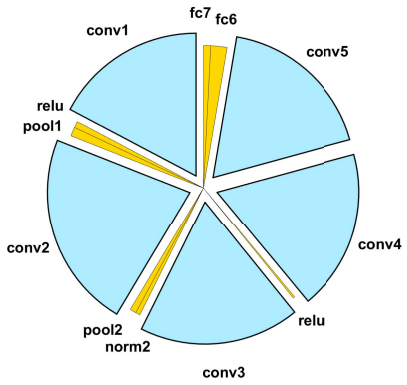

Fig. 1. Computation breakdown for different layers of AlexNet, a popular object detection network. Almost 90% of the computation is dominated by convolution (conv) layers. Rest of the layers, such as ReLU, pooling (pool), and FC, only require 10% of the overall computation.

targeted specifically for deployment in embedded applications where output channel tiling parallelism was exploited. SPARCNet was implemented on an Artix-7 for a VGG-D network and was able to achieve up to 15× improvement in energy efficiency [22] compared with other FPGA-based accelerators while requiring less than 2-W power consumption. Wang *et al.* [29] proposed MINERVA utilizing optimizations, such as fine-grain, heterogeneous data type quantization, dynamic operation pruning, and algorithm-aware fault mitigation for low-voltage SRAM operation and showed 8.1× power reduction on average. Alwani *et al.* [30] proposed fused-layer CNN accelerators which focus on reducing data flow across layers and fusing multiple convolutional layer computation together and achieved 95% reduction in total data transfer. However, not much of research in CNN acceleration on embedded hardware has focused on fast Fourier transform (FFT)-based convolutions (FFT-Convs). Mathieu *et al.* [31] showed fast training and inference of convolutional networks through FFTs in graphics processing unit (GPU) architecture. This was able to achieve significant speed up when a number of feature maps are large. However, for embedded systems with limited memory resources, intermediate memory build up in FFT algorithm can pose a serious issue. Highlander and Rodriguez [32] showed that the CNN can be trained by overlap and add-based FFT transform and potentially avoid the memory augmentation problem and attain 16.3× improvement in computation time over traditional convolution implementation for an 8×8 filter and a 224×224 image. In this paper, we propose FFT-overlap and add method to reduce computations in the convolution layer.

CNNs consist of a variety of layers, such as convolution, FC, max-pooling, batch normalization, and rectified linear unit (ReLU), in which convolution and FC layers are called as weighted layers. Convolution layers are computationally complex due to the sliding window and enormous amounts of multiplications, whereas FC layers are memory-intensive. FFT-Conv is widely used to reduce computation complexity of the convolution layer [33]. However, FFT-Conv is only suitable when data and filter size are similar. When filter size and image size are not matched, it builds up additional intermediate memory to compute FFT coefficients. Thus, FFT-Conv is not a satisfactory option on the cache-limited embedded processor. In this paper, we adopt overlap and add FFT (FFT-OVA-Conv), which overcomes the intermediate memory buildup problem and is suitable for disproportionate data and filter sizes. Furthermore, FFT-OVA-Conv exploits FFT's inherent reduced computational cost than Direct convolution (Direct-Conv) execution.

The key contributions of this paper are as follows:

1) the detailed analysis of FFT-Conv and FFT overlap and add convolution (FFT-OVA-Conv);
2) the analysis of popular CNN networks and choice of network analysis for embedded CNN deployment;
3) the detailed analysis of power-efficient nanoclusters (PENCs) many-core architecture, including postlayout implementation breakdown analysis, memory access architecture, and evaluation methodology;
4) the details of SPARCNet, an FPGA-based CNN accelerator targeted for deployment into embedded applications;
5) implementation of Direct-Conv and FFT-Conv of ResNet-20 on a Zynq FPGA using SPARCNet accelerator, High-level synthesis (HLS) tools, and Verilog;
6) implementation of Direct-Conv, FFT-Conv, and FFT-OVA-Conv techniques for ResNet-20 on PENC many-core, ARM A53 CPU, and NVIDIA Jetson TX1 GPU using CUDA-based 16-bit TensorFlow;
7) thorough cross-platform throughput and timing performance, power, and energy analysis amongst ARM Cortex A53 CPU, PENC many-core, NVIDIA Jetson TX1 GPU, and SPARCNet on Zynq FPGA for different convolution implementation of ResNet-20 on CIFAR-10 data set.

II. Background: Network Layers

In deep CNN architectures, there exists a large variety of layer types, including FC, 1-D/2-D convolutional, pooling, batch normalization, and other specialized forms. There are also activation functions, which are often nonlinear, such as $sigmoid$, $tanh$, and $ReLU$, that can be treated as separate layers. Out of all of the computational layers, FC and convolutional layers are often the most highly utilized in networks and contain the majority of the complexity in the form of computation and memory.

A. Fully Connected Layer

In FC layers, there exists a unique edge between the inputs (or prior layers outputs) and each of the neurons. Each neuron is, therefore, performing a dot product of its inputs with a unique weight vector. The primary issue with FC layers is that for high-dimensional data, the dense connectivity and large parameter set make it difficult to learn a meaningful transformation.

B. Convolution Layer

A convolution (CV) layer can be visioned as an FC layer with added two constraints: the first is that neurons are

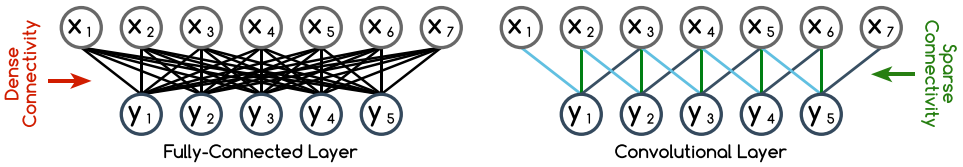

Fig. 2. Side-by-side comparison between an FC layer and a 1-D convolutional layer for input X and output Y. The edges designate multiplication between input and corresponding weight. For convolutional, edge color designates tied weights. In FC, there exists a dense connectivity between inputs and neurons.

connected to only a limited subset of inputs that are in a local neighborhood. For 1-D convolution layers, this corresponds to temporally close inputs, and for 2-D convolution layers, this corresponds to spatially close inputs. The second constraint is that extensive weight sharing is enforced between neurons. These two constraints mathematically correspond to performing a series of convolution operations between the input and set of filters. A convolution layer typically consists of multiple filter banks which we refer to as feature maps. Each feature map is fed to all of the input feature channels that contain temporal/spatial data and produces a corresponding output feature channel. This is achieved by convolving each input channel with a unique filter and summing across the convolved outputs to produce the output feature channel.

Fig. 2 shows a side-by-side comparison of an FC layer and a 1-D convolutional layer. The figure highlights the sparse connectivity obtained using a convolutional layer and the use of weight sharing. In the example, the FC layer requires performing approximately $2\times5\times7 = 70$ operations and storing $7 \times 5 = 35$ weights, whereas the convolutional layer requires performing approximately $2 \times 5 \times 3 = 30$ operations and storing three weights. Convolutional layers can be seen as a form of structured sparsification that significantly reduces complexity while also being able to improve training by reducing the parameter space.

C. Max Pooling Layer

Convolution layers are typically used in conjunction with pooling layers which perform dimensionality reduction by applying a pooling operator, such as average and max, across each input feature channel. Using both of these layers can enable performing feature extraction with desirable properties, such as temporal/spatial invariance. Convolutional and pooling layers can be seen as a form of sparsification and dimensionality reduction that can significantly reduce complexity while being able to better extract features and improve accuracy.

III. Traditional and FFT-Based Convolution Techniques

A. Direct Convolution

Direct-Conv requires a sliding window technique where the filter slides over the data and at each slide position performs multiply-accumulate (MAC) between a data patch and corresponding filter

$$d(n) * f(n) = \sum_{m=-\infty}^{\infty} d(m-n) \times \ f(m)dm. \tag{1}$$

Algorithm 1 FFT-Conv Algorithm

Initialization: Data d with dimension (N,N), Filter f with dimension (K,K)
1. Zero-pad data to the dimension of (N+K-1)
2. Zero-pad filter to the dimension of (N+K-1)
3. Do Fast Fourier Transform of filter to get F
4. Do Fast Fourier Transform of data to get D
5. Multiply to obtain $Y = F \times D$
6. Take Inverse Fourier Transform to get y

For 2-D data with $N \times N$ dimensions and filter of size $K \times K$ ($N > K$), the number of patches can be calculated by $(N - K + 1)^2$ with a single stride. For multichannel data, MAC results between patches of data channels and filter channels are summed to obtain a pixel of the output. Fig. 3(a) shows convolution between a single channel 4×4 data and a 2×2 filter. Nine patches undergo MAC operation to obtain nine pixels of the 3×3 output.

B. FFT-Based Convolution

Convolution in the time-domain transforms into multiplication in frequency-domain

$$d(n) * f(n) = \mathcal{F}^{-1}\{\ \mathcal{F}(d(n)) \times \mathcal{F}(f(n))\}. \tag{2}$$

In this method, first, both filter and data are transformed into the frequency domain by the FFT of (N + K − 1) length. Therefore, filter FFT coefficients require additional intermediate memory. Then, data FFT and filter FFT are elementwise multiplied and fed to an inverse FFT (IFFT) to obtain the output as shown in Algorithm 1. Fig. 3(b) shows that the filter dimension of 2×2 is augmented to 5×5 due to initial FFT transform.

C. Overlap and Add FFT Convolution (FFT-OVA-Conv)

The overlap and add FFT method takes a portion of data and treats it as an independent input for convolution with the filter. This means that data [i.e., $d(n)$] can be segmented into chunks of $d(n - kL)$, where L is the length of each segment. FFT-Conv is performed on these segmented data and then block outputs are aligned and added as showed in (3) and explained in Algorithm 2

$$d(n) \star f(n) = \sum_{k} (\mathcal{F}^{-1}\{\mathcal{F}(d(n-kL)) \times \mathcal{F}(f(n))\}). \tag{3}$$

Algorithm 2 FFT-OVA-Conv Algorithm

Initialization: Data d with dimension (N,N), Filter f with dimension (K,K)

1. Segment data D into non-overlapping blocks of KxK dimension
2. Zero-pad filter to the dimension of (K+K-1)
3. Do Fast Fourier Transform of filter to get F
4. **for** *each* O_{ij} *data block* **do**
   - Zero-pad blocks to the dimension of (K+K-1)
   - Do Fast Fourier Transform of blocks to get I_{ij})
   - Multiply to obtain $Y_{ij} = F \times D_{ij}$
   - Take Inverse Fourier Transform to get y_{ij}

end

5. Obtain y by overlap and adding last (K-1) columns of y_{ij} with first (K-1) columns of y_{i+1j}; and overlap and adding last (K-1) rows of y_{ij} with first (K-1) rows of y_{ij+1}

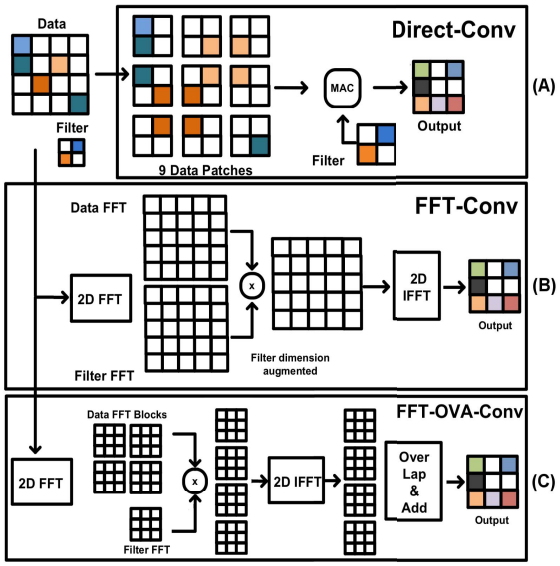

Fig. 3. Different convolution schemes. (A) Direct-Conv with computational complexity of $O(N^2\ K^2)$. (B) FFT-Conv with computational complexity of $O(N^2 \log(N))$. (C) FFT-OVA-Conv with $O(N^2 \log(K))$ where data dimension is $N \times N$ and filter dimension is $K \times K$.

Fig. 3(c) shows a 4×4 data to be segmented into four blocks of 2×2 size (3×3 after zero-padding) which corresponds to the filter dimension. Each block undergoes FFT-Conv in the next stage. Minimal additional intermediate memory storage is created in this scheme, since the inputs to FFT are of the same dimension. The outputs of FFT-Conv from four blocks of size 3×3 are aligned and added to obtain the 3×3 output.

IV. Network Analysis

A. Computational Complexity and Memory Access Analysis

In terms of computational complexity, Direct-Conv between $N \times N$ image and $K \times K$ filter kernel requires $(N - K + 1)^2 \times K^2$ multiplication operations. The order of computation complexity is $O(N^2\ K^2)$. Depending upon stride,

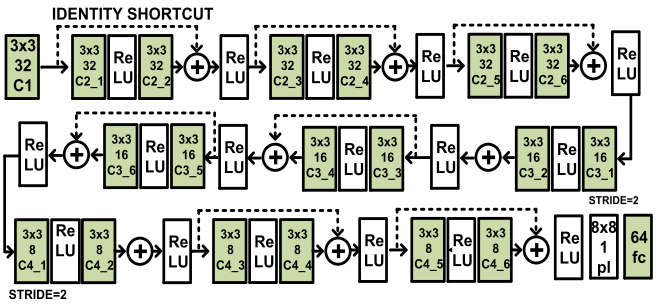

Fig. 4. ResNet-20 with CIFAR-10 data set architecture. C: convolutional layer with the first row as filter size and second row as dimensions of output. ReLU: rectified linear unit. pl: global average pooling layer with the first row as window size and second as output size. fc: fully connected layer.

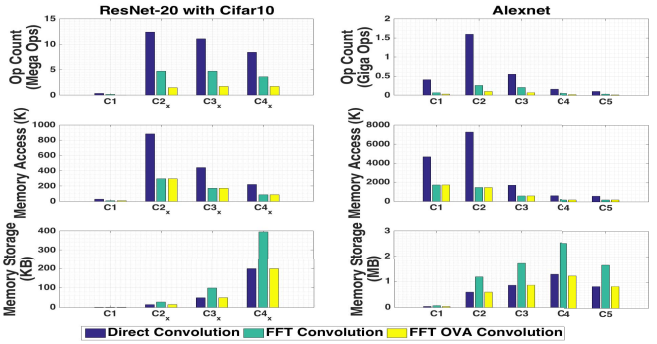

Fig. 5. Computational complexity and memory analysis of three convolution methods for ResNet-20 and AlexNet. ResNet-20 has a deeper network of 19 convolution layers, AlexNet has a relatively shallow network of five convolution layers. C1_x, C2_x, and so forth represent groups of six similar convolution layers in ResNet-20. FFT-OVA-Conv reduces computation cycles 7× and 10× than Direct-Conv and 2.8× and 2.5× than FFT-Conv. FFT-Conv is on average up to 3× better than Direct-Conv. Intermediate memory storage for FFT-Conv requires 2× more space than Direct-Conv and FFT-OVA-Conv.

one memory location in the image is accessed K times, and the memory storage requirements are $N^2 + K^2$. FFT-Conv requires $6CN^2 \log(N) + 4N^2$ operations where C is a constant, with a computation complexity of $O(N^2 \log(N))$. However, memory storage requirements are $2N^2$, since the filter undergoes a higher point FFT and require additional space. In FFT-OVA-Conv, computational complexity is reduced to $O(N^2 \log(K))$, and there are minimal additional memory storage requirements compared with Direct-Conv.

We evaluate computational complexity and analyze memory requirements for ResNet-20 [12] architecture with CIFAR-10 and AlexNet [1]. CIFAR-10 data set contains 50k training images and 10k testing images of 32×32 size. ResNet for CIFAR-10 can be represented by $6n + 2$ stacked weighted layers with a global average pooling and softmax at the end. Only identity shortcuts were used in this network. We take $n = 3$ which results in ResNet-20 as shown in Fig. 4. AlexNet consists of five sequential convolutional layers with three different filter sizes of 11×11, 5×5, and 3×3. All filters in ResNet-20 are 3×3. Fig. 5 shows that for both AlexNet and ResNet-20, FFT-OVA-Conv reduces computation 7× and 10× compared with Direct-Conv and 2.8× and 2.5× compared with FFT-Conv which is evident in initial layers

TABLE I
PARAMETER COMPLEXITY AND ACCURACY OF POPULAR OBJECT DETECTION NETWORKS (CV = CONVOLUTION AND FC = FULLY CONNECTED)

Network	AlexNet	VGG-B	GoogLeNet	VGG-D	Inception-V3	ResNet-20	Human
Year	2012	2014	2014	2015	2015	2015	
CV Layers	5	10	21	13	77	19	
FC Layers	3	3	1	3	1	1	
CV Params	2.3M	9.2M	4.8M	14.7M	12M	0.27M	
FC Param	57.5M	123.6M	1M	123.6M	2M	.1K	
Top-5 Error	15.30%	9.60%	9.20%	7.20%	5.60%	8.75%	5.10%

where input image size is larger than filter kernel. These large improvements come from the fact that unlike FFT-based convolutions (FFT-Conv and FFT-OVA-Conv), Direct-Conv requires a sliding window technique to perform convolution. This technique makes it computationally more expensive than FFT-OVA-Conv. In the sliding window approach, a filter kernel slides over data and at each position performs a dot product between the kernel and patch of data for each output pixel. For example, with a 3×3 kernel sliding over a 9×9 input, there will be $3 \times 3 = 49$ positions the filter kernel will slide too. This will require 17 Ops (per slide position) $\times$ 49 (number of slide position) operations. On the other hand, FFT-OVA-Conv simply segments the 9×9 data into nine blocks of 3×3 size and perform the FFT. This requires 7 Ops $\times$ 9 (number of segments) + overheard of FFT/IFFT.

In terms of memory access, both the types of FFT-Conv are on average 3$\times$ better than Direct-Conv, while intermediate memory storage for FFT-Conv requires 2$\times$ more space than Direct-Conv and FFT-OVA-Conv.

B. Choice of Network for Embedded Devices

For the scope of this paper, we target embedded devices and tiny cores with memory constraints; therefore, the amount of filter weights or convolution parameters needs to be small enough for storage or streaming to and from embedded devices and tiny cores, while networks or data sets that require a significant amount of memory storage and transmission-embedded devices with a larger memory will be a better choice. For example, for CNN deployment in embedded devices and tiny cores, the networks that require a lower number of parameters without sacrificing considerable accuracy are preferred. Table I shows the details of popular object detection networks, such as AlexNet [34], VGG-B [35], GoogLeNet [11], VGG-D, Inception-V3 [36], and ResNet20 [12]. In recent years, networks have more convolution (CV) layers and only one FC layer at the end of classification. For example, Inception-V3 has 77 CV layers and 1 FC layer, whereas previous predecessors such as AlexNet consist of only five CV layers but three FC layers. The amount of filter weights or convolution parameters needs to be small enough for storage or streaming in embedded devices and tiny cores with memory restraints. In this respect, ResNet-20 showed in Fig. 4 consisting of 19 CV layers and 1 FC layer which requires 0.27M CV parameter which is a good candidate for embedded CNN deployment. Second, the error rate must not be compromised when choosing networks with lower CV parameters. ResNet-20 has an 8.75% error rate which is 1.7$\times$ better than AlexNet and almost similar to human efficiency. Thus, in this paper, ResNet-20 was considered for the evaluation of CNN and FFT techniques on the embedded platforms.

V. EMBEDDED PLATFORMS AND EXPERIMENTAL SETUP

A. PENC Many-Core Overview and Key Features

A PENC many-core accelerator is a homogeneous architecture that consists of in-order tiny processors with a six-stage pipeline, an RISC-like DSP instruction set, and a Harvard architecture model [37], [38]. The core operates on a 16-bit data-path with minimal instruction and data memory suitable for task-level and data-level parallelism. Furthermore, these cores have a low complexity and minimal instruction set to further reduce area and power footprint. The lightweight cores also help to ensure that all used cores are fully utilized. The processor can support up to 128 instructions, 128 data memory, and provides 16 quick-access registers. In the network topology, a cluster consists of three cores that can perform intracluster communication directly via a bus and intercluster communication through a hierarchical routing architecture. Each cluster also contains a shared memory. Fig. 6 shows the block diagram of a 16 cluster version of the design, highlighting the processing cores in a bus-based cluster. Each core, bus, shared memory, and router was synthesized and fully placed and routed in a 65-nm CMOS technology using Cadence System-on-Chip (SoC) encounter and results for one cluster using the tools postlayout results are summarized in Fig. 6(e). The tiny processing core contains additional buffering on the input in the form of a 32-element content-addressable memory (CAM). It is used to store packets from the bus and allow a finite state machine (FSM) to find a word where the source core field corresponds to that in the IN instruction itself, where the IN instruction is used to communicate between the cores. For example, if the core is executing `IN 3`, the FSM searches through the CAM to find the first word whose source core is equal to three. This word is then presented to the processing core and processing continues. Our initial many-core architecture design had four processing cores without shared memory, which was ideal for DSP kernels with minimal data storage. Since CNNs require a large amount of memory for their model data, the PENC many-core architecture replaces the fourth core with a shared memory to accommodate the memory requirements. Our initial results showed that performance benefit of bringing additional cores within the cluster diminishes given the increase in total area, power consumption, and network congestion. The key

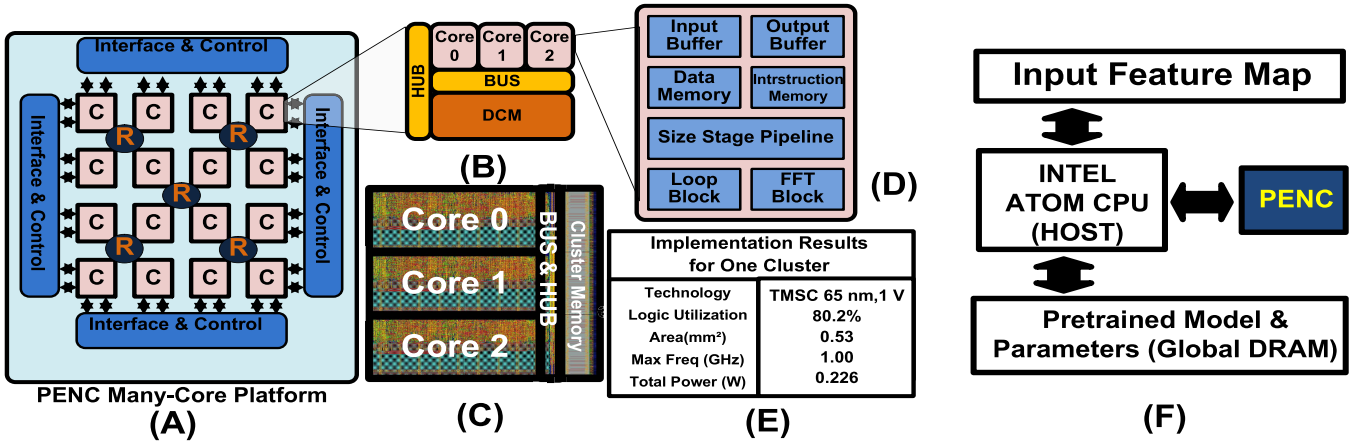

Fig. 6. (A) PENC many-core architecture with 16 cluster example. (B) Bus-based cluster architecture. (C) Postlayout view of a cluster implemented in 65-nm, 1-V TSMC CMOS technology. (D) Single-core architecture with FFT block. (E) Postplace and route implementation results of a cluster (consisting of three cores + bus + CM) at 65-nm CMOS technology. (F) Heterogeneous deployment of PENC with Intel Atom CPU (host) where pretrained model and parameters (filter weights) are used for network inference.

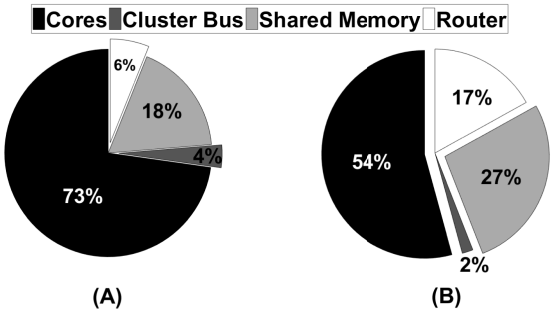

Fig. 7. Postlayout implementation breakdown analysis of PENC many-more comprising of 192 processing cores, cluster bus, shared memory, and router. (A) Area breakdown. (B) Power breakdown.

characteristics of the PENC many-core platform are discussed in the following.

1) Domain-Specific Customization of Instruction Sets: The PENC architecture is optimized for machine learning kernels [37]. These are lightweight processing cores containing a limited instruction set for efficiency with a handful of specialized instructions, such as FFT. PENC has 8 and 16 point built-in FFT instruction. The FFT instruction activates the hardware block that provides the FFT addresses for a complex radix-2 FFT. Upon the source and destination specified, the address in memory containing the real and imaginary values for inputs is retrieved from the hardware block. Fig. 7 shows the postlayout implementation breakdown analysis of optimized PENC many-core comprising of 192 Processing cores, bus cluster, shared memory, and router with Fig. 7(a) showing the area breakdown and Fig. 7(b) showing the power breakdown. These results are obtained by place and route using Cadence Encounter for the 65-nm technology. The area results come from the postlayout report, and the power results are obtained from the encounter power analysis with careful consideration of activity factor, capacitance, IR drops, and rail analysis. These results are used to compare with the off-the-shelf processors.

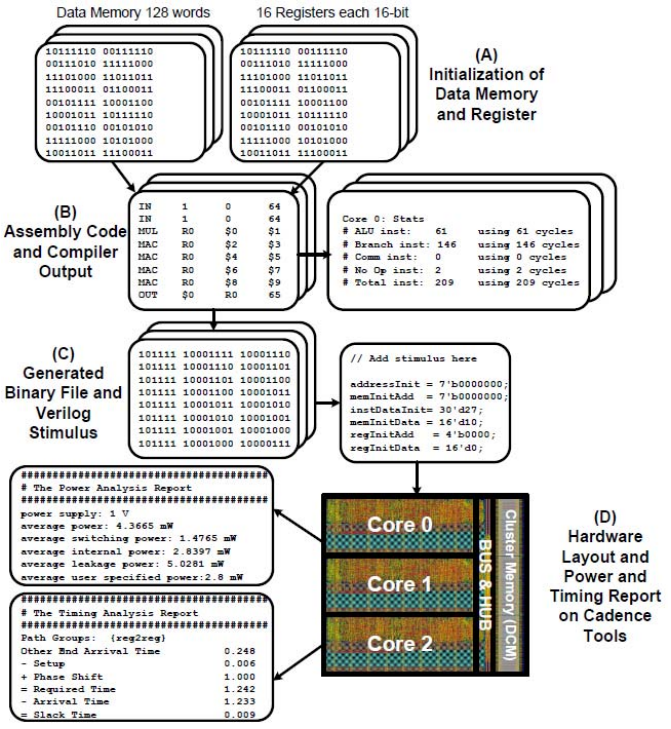

Fig. 8. PENC many-core cycle accurate simulator and compiler flow for applications written in assembly. The simulator uses the PENC VLSI hardware statistics from Cadence SoC postlayout results.

2) Efficient Cluster Memory Access Architecture: While the lightweight cores are ideal for DSP kernels that require minimal static data [39], [40], and ML kernels often require larger amounts of memory for their model data. This is addressed with the distributed cluster-level shared memory that is interfaced to the bus. The shared memory within a cluster consists of three instances of SRAM cells of memory size 1024×16 bits making up a total of 3072 words and can be accessed within the cluster using the bus and from other clusters through the router. To access the memory, cores use

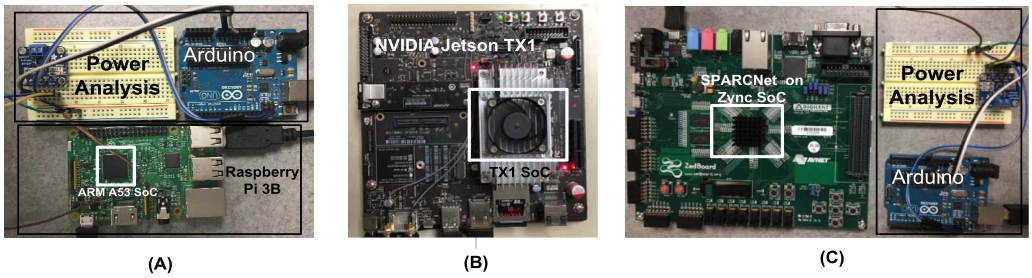

Fig. 9. (A) ARM Cortex A53 CPU residing in Raspberry Pi 3 B platform with power measurement setup. (B) NVIDIA Jetson TX1 GPU platform with integrated power measurements. (C) SPARCNet accelerator on Zynq 7020 FPGA with power measurement setup.

two memory instructions: `LD` and `ST`. The maximum depth of the cluster memory (CM) is 2^{16} words, since registers and data memory are both 16-bit wide and can, therefore, supply a 16-bit memory address. Using data memory as operands for instructions is still beneficial to using `LD` and `ST` from an efficiency standpoint because of the one-cycle read/write capability.

3) PENC Platform Evaluation Setup: For evaluation of applications running on the PENC many-core, we developed cycle-accurate simulator and compiler that take user's code and the postlayout hardware results, as shown in Fig. 8. Careful attention was paid to this hardware simulator for a fair comparison. The simulator provides cycle accurate results, including completion time, instructions, and memory usage per core. It also serves as a reference implementation of the architecture; its purpose is to make testing, refining, and enhancing the architecture easier. Each task of the algorithm is first implemented in assembly language on every processing core using the many-core simulator. The simulator reads in the code as well as initializes the register file and data memory in each core. It then models the functionality of the processor and calculates the final state of register files and data memories. Binary files generated by the many-core compiler are used to program each core individually. For execution time and energy consumption analysis of the algorithm, binaries obtained from many-core compiler are mapped on to the hardware design of the many-core platform (in Verilog) and simulated using Cadence NC-Verilog as shown in Fig. 8. The activity factor is then derived and is used by the Cadence Encounter tool for accurate power estimation of application. The many-core simulator reports statistics such as the number of cycles required for arithmetic logic unit, branch, and communication instructions, which are used for the throughput and energy analysis of the PENC many-core architecture. For power evaluation and comparison with the other systems, we have included the power and latency of using the PENC with a host processor, an Intel Edison. In the evaluation, we have added the Intel Edisons power consumption when idle and active. Power consumption for Edison is found using an INA219 for power measurement. While the latency for cache and memory for the Edison is benchmarked using the open-source tool Calibrator [41]. Each of these, power and latency, is added to the PENC numbers when in comparison against the other platforms.

B. COTS Devices Setup

For the commercial off-the-shelf platforms, we disabled unnecessary board accessories, such as WIFI, or HDMI, and disconnected all external accessories other than Ethernet.

C. ARM Cortex-A53 CPU

The Cortex-A53 is equipped with 32- and 64-bit instruction sets and eight-stage pipeline, efficient data fetch, and access mechanism to maximize performance and low power. The Cortex-A53 CPU residing on Raspberry Pi 3 B is evaluated for energy and power consumption by collecting the current consumed at the board level as shown in Fig. 9(a). The power consumption of the entire system was captured using a TI INA219 voltage and power IC sampled by an Arduino Uno as shown in Fig. 9(a). The average active current is used in calculations for the platform as is the average current when the platform is executing code.

D. NVIDIA Jetson TX1 GPU

The NVIDIA Jetson TX1, as shown in Fig. 9(b), is an SoC combining the 256-core Maxwell GPU and an ARM Cortex-A57 processor. The A57 processor configuration consists of four ARM Cortex-A57 processors. Each ARM A57 CPU has a 48-KB L1 data and 32-KB instruction cache supporting 128-bit NEON general-purpose SIMD instructions. All processors in the configuration have shared access to a 2-MB L2 cache. Power monitoring for this platform comes from the TX1's bus addressable discrete current/power monitor. For accuracy, the average power during execution is used for the power and energy computations. For the network development and testing, we use Torch, a scientific computing framework which provides an efficient implementation of the models on the CPUs and embedded GPU. By exploiting the GPU, we are able to achieve several orders of magnitude energy-efficiency improvement over the ARM CPU counterpart.

E. SPARCNet: FPGA Accelerator on Zynq

SPARCNet, is a configurable FPGA hardware-based accelerator for SPARse Convolutional NETworks and was developed by our group [22]. The accelerator was designed to efficiently deploy any convolutional networks in embedded real-time applications that have limited power and area budgets. The SPARCNet is built using both Xilinx's Vivado

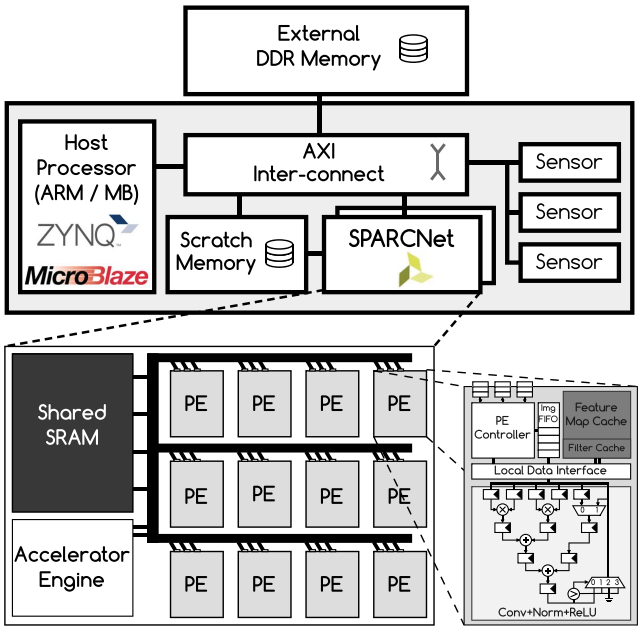

Fig. 10. High-level hardware block diagram containing the SPARCNet accelerator, consisting of the host processor, SPARCNet accelerator, memory, and AXI interconnect backbone. The PEs are arranged in a grid with a unified network interface. The network provides two sets of buses that are unidirectional and contain two 16-bit channels. A shared on-chip memory is used to cache the current layers feature channels along with layer parameters.

HLS language, for convolution, batch normalization, ReLU, and Verilog. Specifically, it targets accelerating convolutional layers, since the high percentage of computational (> 90%) resources are required. Modifications to SPARCNet's base configuration are done using a combination of C++ and HLS. The SPARCNet comes with a host C++ code as application programming interface to redefine a model, weights and layers. Modification of hardware acceleration features, such as a new layer type, requires the modification of the Xilinx HLS files. The high-level block-diagram of the accelerator is depicted in Fig. 10 which consists of five main parts: host processor, communications bus, memory, accelerator engine (AE), and processing element.

The SPARCNet depends on a host processor, such as ARM Cortex-A9, tasked with serializing and dispatching a predefined network topology layer-by-layer. It contains two communication buses, an AXI bus, and SPARCNet's internal dual-channel 16-bit bus. The AXI bus provides the main interconnection and communication between the host processor and SPARCNet and has shared access to the external DDR3 RAM. The internal dual-channel 16-bit bus is used for command and control from the AE, processing engine (PE) communications, and access to the shared SRAM. For memory, there is a shared, between the host and SPARCNet, DDR3 bank of memory. The PEs also have a shared SRAM that is used for caching current layer features and parameters. The AE is used to communicate with the host and to configure, as well as, to perform data-marshaling for the PEs. Communication between the AE, host, and external DDR memory is done through the AXI interface.

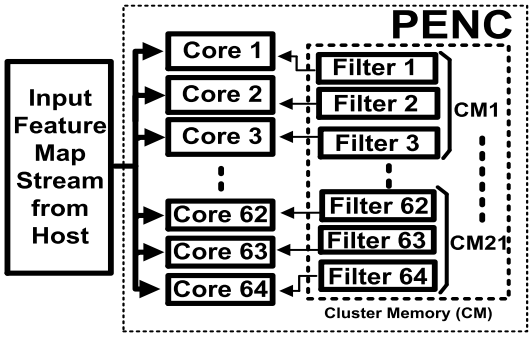

Fig. 11. Data flow between cores with host streaming input feature maps to cores. Filter weights are saved in CM in PENC.

The PEs are used to perform concurrent processing by parallelizing across the output channels/feature maps. Each PE contains a local scratchpad memory to store both its current filter as well as its partial output channel result. This allows quickly adding the convolved output of consecutive input channels to the local result and only performing write back to shared memory after the final input channel. All computation and storage are done using 16-bit floating point numbers which includes the benefits from standard floating-point numbers, including dynamic range, underflow/overflow conditions, and a nonuniform scale while requiring fewer resources.

The accelerator also supports sparsification by containing both a filter and input channel bit vector mask to define connectivity maps between input channels and feature maps as a configuration option. This enables omitting pruned filters and unnecessary convolution operation between zeroed input channels and/or corresponding filters. Another key feature of the SPARCNet is the fusion of convolution, batch normalization, and ReLU layers, with bypass options, into a single operation. These three layers are often found in succession within the CNN, and the fusion of these can help to significantly reduce wasted memory transfers and operations. Fig. 9(c) shows the power measurements setup for the SPARCNet on the Zynq platform with the use of an external current/power sensor and an Arduino microcontroller.

VI. Implementation Results

A. PENC Many-Core Implementation

ResNet-20 with CIFAR-10 is taken as a case study for evaluating convolution performance on PENC many-core, NVIDIA Jetson TX1, ARM A53 CPU, and SPARCNet. Fig. 6(f) shows the heterogeneous platform consisting of PENC as a convolution accelerator and Intel Atom CPU as host performing task scheduling, data marshaling, and postprocessing. Pretrained model parameters are stored on the global DRAM shared between a host processor and a PENC accelerator. The double buffering approach is adopted to hide data memory transfers between PENC and Atom. Inactive cores in PENC are shut down by power gating to reduce power consumption.

Instruction-level parallelism (ILP) is achieved by broadcasting input feature maps to active cores to perform convolution with different filter weights stored in corresponding CMs

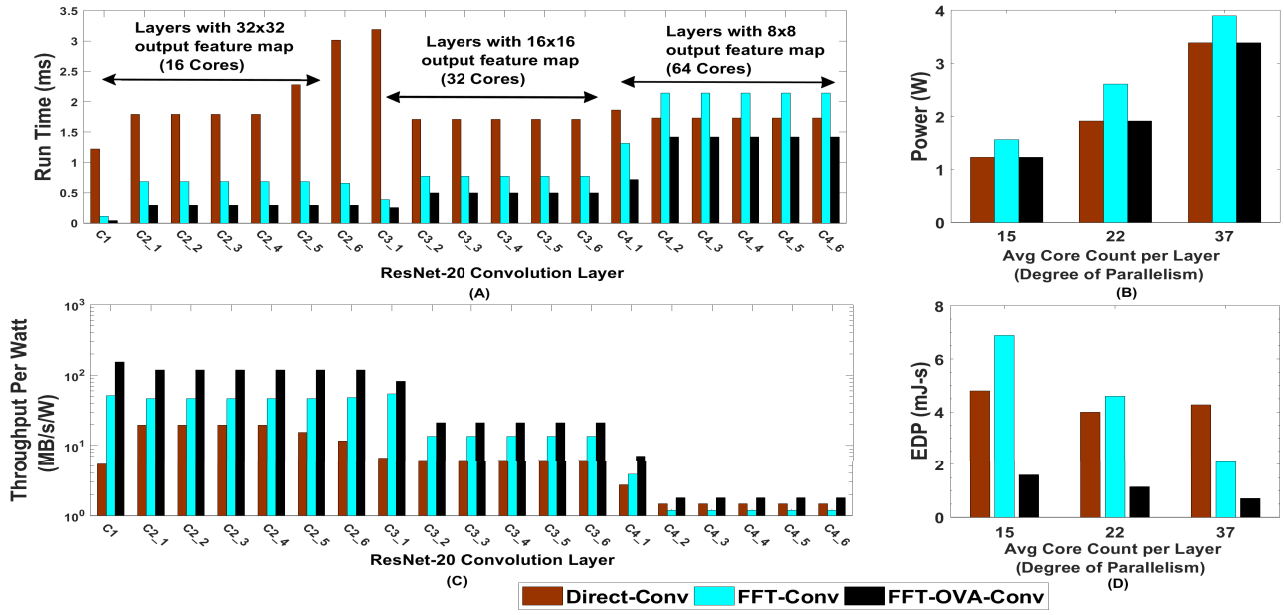

Fig. 12. (A) Layerwise run time breakdown for most-parallel implementation on PENC. (B) Total power for PENC. (C) Layerwise throughput per watt for most-parallel implementation on PENC. (D) EDP for three different implementations on PENC: semiserial implementation with 15 cores/layer, semiparallel implementation with 22 cores/layer, and most-parallel implementation with 37 cores/layer.

as shown in Fig. 11. Three different levels of parallelism are implemented based on storage and scope of ILP: semiserial, semiparallel, and most-parallel. In a semiserial implementation, a number of cores are assigned as per a minimum number of cores required to store total layer filter weights, which is on average 15 cores per layer for ResNet-20. In the most-parallel implementation, core number is assigned equal to the number of filters, thus achieving maximum ILP which results in 36 cores per layer on average. FFT-Convs are implemented with PENC many-core FFT instruction. The 3×3 filters are zero padded to either 8 or 16 dimensions as per the size of the data. If data dimension is the same or higher than 16 such as in $C1, C2\_x, C3\_x$ layers, 16 point FFT is used. Here, $C1$ represents the first convolution layer, $C2\_x$ represents the second layer group with 32×32 data dimension, and so forth. The 2-D FFT is performed by first finding 1-D FFT of data in the x-direction followed by 1-D FFT in the y-direction. Since intermediate memory is created in FFT-Conv after first FFT, we use CM from a nearby idle cluster for additional storage. Therefore, each cluster is surrounded by a storage cluster where cores are idle but CM, and corresponding routers are used. FFT-OVA used only eight-point FFT operation as data are segmented into 3×3 blocks.

FFT-OVA-Conv improves total run time by $2.9\times$ and $1.65\times$ than Direct-Conv and FFT-Conv. Fig. 12(a) shows the layerwise run time breakdown for most parallel implementation. The memory storage requirement in the last five layers of ResNet-20 is significantly larger than previous layers (see Fig. 5). Since CM in PENC is limited to 6 KB, for the last five layers of FFT-Conv implementation, we have to use memory from adjacent clusters to facilitate additional storage requirement. Intercluster communication latency in PENC is almost $2\times$ higher than intracluster communication. Therefore,

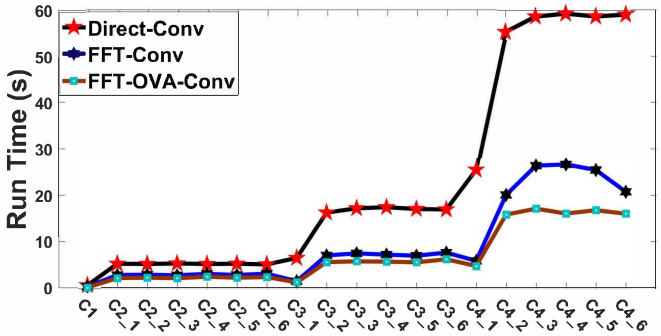

Fig. 13. Layerwise run time for ARM Cortex-A53 CPU running at 1.2 GHz for serial implementation of ResNet-20 with CIFAR-10 network for different convolution methods.

in Fig. 12(a), in last five layers, the implementation for FFT-Conv has the highest run time than Direct-Conv and FFT-OVA-Conv. Power increases in FFT-Conv by $1.4\times$ than both counterparts due to additional router and storage CM power as shown in Fig. 12(b). Throughput per watt and energy delay product (EDP) improve in FFT-OVA-Conv by $6.8\times$ and $2.5\times$, and $6\times$ and $2.9\times$, respectively, than Direct-Conv and FFT-Conv. Detailed results are tabulated at Table II.

B. ARM Cortex-A53 CPU Implementation

We implemented the ResNet-20 on the Raspberry Pi 3 Bs low-power ARM Cortex-A53 CPU running at 1.2 GHz using existing software libraries available for Linux. The implementation is written in GNU Octave [42] high-level programming language. Here, we created layers to support the ResNet-20 model with user-defined serial functions for convolution, ReLU, and average pooling operation. Our decision

TABLE II
COMPARISON OF RESNET-20 IMPLEMENTATIONS FOR CIFAR-10 DATA SET ON PENC

Platform	Method	Execution Time (ms)	Energy (mJ)	Throughput Per Layer (MB/s)	EDP (mJ-s)
PENC (Semi-serial)	Direct-Conv	66.5	72	5	4.8
	FFT-Conv	67.2	103	6	6.9
	FFT-OVA-Conv	31	52	18	1.6
PENC (Most-parallel)	Direct-Conv	36	119	10	4.2
	FFT-Conv	21	103	30	2.1
	FFT-OVA-Conv	12	57	60	0.7

to use GNU Octave comes from previous experience with Octave, the ability to run the framework directly on the target devices, and from our experience with the performance being on par with that of MATLAB. The timing results reported for the platform are obtained from Octave's time measurement function. Fig. 13 shows the layerwise timing breakdown for three different convolution methods. FFT-OVA-Conv achieves 3.36× and 1.38× improvement in execution time and 2.72× and 1.32× better throughput than Direct-Conv and FFT-Conv.

C. NVIDIA TX1 Jetson GPU Implementation

For the ResNet-20 with CIFAR-10 data set implementation, the embedded GPU is used as an efficient accelerator. The data set was trained using Torch7, a scientific computing framework, with the CUDA 7.0/8.0 release, including cuFFT, cuBLAS, and cuDNN v4 and v5.1. For the model, we configured Torch7 to use half float tensors, enable the cuDNN backend, and set the cuDNN to its fastest model, to enable the use of FFTs. To be consistent with the other implementations, we limited the batch size to 1. The model for ResNet-20 is trained on NVIDIA GTX-1080 for the CIFAR-10 data set. The inference learning is performed on NVIDIA TX1 configured with dual active ARM cores, running at 1.7 GHz and a GPU configured for 998.4 MHz. For this paper, we implemented the ResNet-20 using both Direct-Conv and FFT-Conv. FFT-Conv is 1.9× faster and 2.2× more energy-efficient and achieves 5.6× better throughput per layer than Direct-Conv. For the libraries used for FFT-Conv and FFT-OVA-Conv and the small data size, >6 KB per max image, the overhead associated with launching multiple concurrent FFTs and IFFTs outweighs the potential speed up.

D. SPARCNet on Zynq FPGA Implementation

The SPARCNet hardware accelerator is reconfigured to implement a ResNet-20 network with the CIFAR-10 data set using both Direct-Conv and FFT-Conv techniques and is deployed onto the Zynq 7020 FPGA. In both the Direct-Conv and FFT-Conv configurations, the designs use 16-bit floating number data representation, a 154-MHz FPGA clock, and a 700-MHz ARM processor, and the convolution, normalization, and ReLU operations are implemented in the FPGA fabric. While the ARM Cortex-A9 is used for serialized actions for all other layer functions in ResNet. In the case of Direct-Conv, the implemented design is either using 1, 16, or 32 PEs.

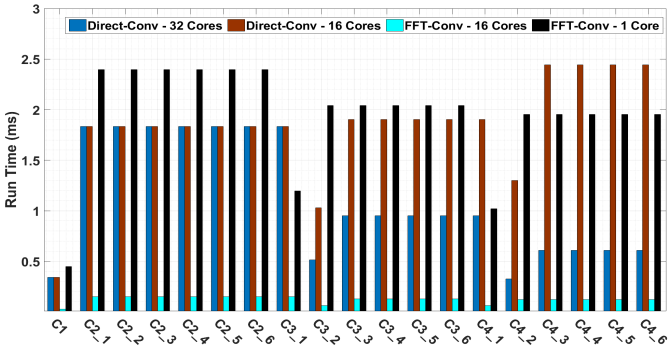

Fig. 14. Layer execution time analysis for SPARCNet with 32, 16, and 1 PEs active for Direct-Conv and 16 or 1 PE for FFT-Conv.

For the FFT-Conv design, we used the Xilinx generated FFT core in the PEs. While this core is efficient, the design limited the number off PE FFTs due to FPGA resources. Only two different configurations were implemented, 1 or 16 1D FFT PEs. In this case, all FFTs are used to calculate both FFT and IFFT, and the PEs still provide normalization and ReLU in the FPGA. As in the Direct-Conv case, other layer functions are executed in software on the ARM.

Fig. 14 presents the layer-by-layer timing analysis for a different number of active processing elements in the SPARCNet design. SPARCNet is bound by the number of active processing elements even as the total data size decreases. The only one not to show this trend is the Direct-Conv with 32 PEs and the 16-PE FFT-Conv Design. The Direct-Conv with 32 PEs has the classic stair step decrease, while the FFT-Conv with 16 PEs the time is consistent between layers due to the limited resources and constant sized FFTs with zero padding used on all successive layers. With that explanation, we still have almost a 6× decrease in execution time when using the FFT-Conv, compared with the Direct-Conv on SPARCNet. Fig. 15 shows the complete execution time for three different PE configurations. The first is the execution time for a 32-PE design, but due to limited FPGA resources, only the Direct-Conv method is presented. The second grouping of bars is the 16 PE designs, and the final is the one-PE design.

ResNet-20, compared with earlier works [22], has reduced requirements for memory. This reduction helps the implementation gain further speed up with being able to fully utilize the shared SRAM and in turn reduce the dependence on the external DDR memory for retrieval of layer weights and image

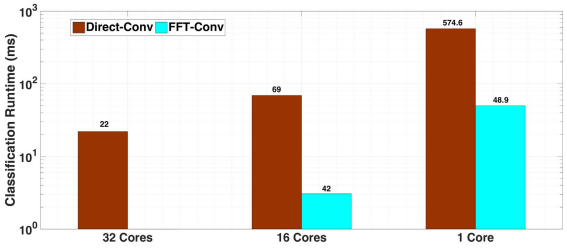

Fig. 15. Total execution time using SPARCNet configured with 32, 16, or 1 PE(s) for Direct-Conv and 16 or 1 PE(s) for FFT-Conv. There is no bar for 32 PEs in the FFT-Conv configuration as there are not enough FPGA resources on the Zynq 7020 to support that configuration. Please note that the scale on this graph is log-based.

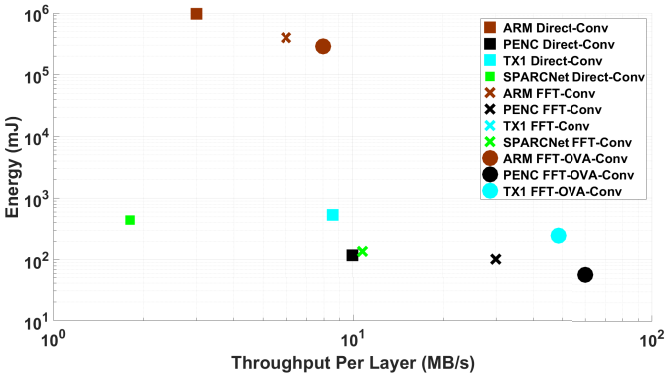

Fig. 16. Cross platform energy efficiency and throughput per layer analysis for ResNet-20 between ARM Cortex-A53 CPU, PENC many-core, NVIDIA TX1 GPU, and SPARCNet FPGA.

caching. These changes enabled the SPARCNet Direct-Conv implementation to have a total run time of 220 ms for one image with 1.6-MB/s throughput per layer and 443-mJ energy consumption, whereas the FFT-Conv, configured with 16 1-D FFTs, had an execution time of 42 ms with 137-mJ energy consumption and a throughput per layer of 10.8 MB/s.

E. Cross Platform Analysis

In this paper, the software implementation consists of general purpose embedded systems PENC, ARM A53 CPU, and NVIDIA TX1 GPU, where we program the cores/threads to map our CNN algorithms and we exploit the parallelism across these architectures. The hardware implementation includes reconfigurable SPARCNet accelerator on a Zynq FPGA which is specifically implemented for a CNN. Across three different general purpose software platforms, PENC is 10916× and 1.8× faster than ARM A53 CPU and NVIDIA Jetson TX1 GPU, respectively, as shown in Fig. 17. In terms of energy, PENC is 5053×, 4.3×, and 2.4× more energy-efficient than ARM A53 CPU, NVIDIA Jetson TX1 GPU, and SPARCNet. In terms of throughput per layer, PENC shows 7.5× and 1.2× improvement over ARM A53 CPU and NVIDIA Jetson TX1 GPU, respectively. Fig. 16 shows energy efficiency versus throughput per layer across and between different platforms. In this figure, the best implementation will exhibit higher throughput per layer at lower energy consumption. Among general purpose software platforms, PENC with

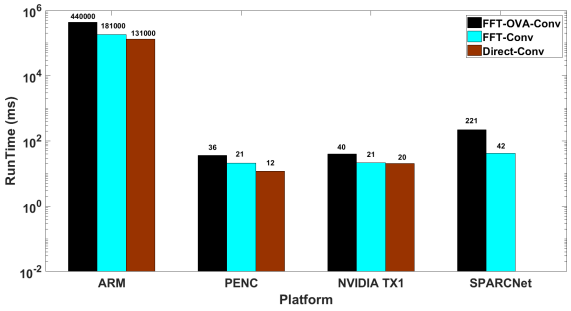

Fig. 17. Cross platform runtime comparison for ResNet-20 between convolution methods on ARM A53 CPU, PENC many-core, NVIDIA TX1 GPU, and SPARCNet FPGA with 32 PEs for Direct-Conv and 16 PEs for FFT-Conv.

FFT-OVA-Conv implementation has the highest throughput with least amount of energy consumption among TX1,ARM A53, and SPARCNet implementations. In terms of FFT-Conv implementation between PENC, TX1 GPU, and SPARCNet, TX1 shows 1.6× better throughput with the expense of 4.9× increased energy consumption, whereas the SPARCNet was 1.08× higher in throughput but 2.84× higher in energy consumption.

F. Comparison With Others

Although there are several works on CNN implementation, we could not find an implementation for ResNet-20 on the CIFAR-10 data set. Ma [43] showed end-to-end scalable FPGA accelerator for ResNet-50 and ResNet-152 on Altera Arria-10 GX1150 FPGA with the 20-nm technology and achieved 27.2 and 71.71 ms of execution time with 16-bit data precision and 69% DSP and 80%–93% RAM utilization. Our implementation for ResNet-20 in SPARCNet on Zynq FPGA achieved 129–341× faster execution time than their work without scaling.

Shen [44] showed hand-designed features fed to a ResNet-200 for a specific task of pedestrian detection which achieved 24-ms run time in Jetson TX1 GPU with 64M parameters which are 1.12× better than our implementation in TX1 GPU without scaling. This arises from the fact that, during inference, we restricted the batch size to one image to keep similarity between implementations on the rest of the platforms, such as PENC, ARM A53 CPU, and SPARCNet. Using bigger batch size will pipeline the computation and reduce overall setup time in TX1 and can achieve significant speedup.

VII. Conclusion

Three convolution techniques, namely, Direct-Conv, FFT-Conv, and FFT-OVA-Conv, were explored on a low-power domain-specific many-core architecture named PENC, NVIDIA Jetson TX1 GPU, ARM Cortex A53 CPU, and SPARCNet on Zynq 7020 FPGA to explore the tradeoff within software and hardware implementation, domain-specific logic and instructions, as well as different parallelism across different architectures. These three techniques were implemented and evaluated on general purpose software

platforms, including an ARM A53 CPU, PENC many-core, using a built-in FFT instruction, and the NVIDIA Jetson TX1 GPU, for ResNet-20 with CIFAR 10 data set with a variety of parallel mappings. In addition, Direct-Conv and FFT-Conv were implemented on Zynq 7020 FPGA using a CNN hardware accelerator named SPARCNet. PENC FFT-OVA-Conv has an execution time of 12.4 ms and average per layer throughput of 60.5 MB/s with 3.38-W power consumption and is 2.9× and 1.65× faster and achieves 6.8× and 2.5× higher throughput per watt than Direct-Conv and FFT-Conv, respectively. Implementation on SPARCNet on Zynq FPGA achieves the execution time of 42 ms with 10.8-MB/s throughput per layer with 142-mJ energy consumption for FFT-Conv. On the ARM A53 CPU, the FFT-OVA-Conv achieves 3.36× and 1.38× improvement in execution time and achieves 2.72× and 1.32× higher throughput than Direct-Conv and FFT-Conv. On NVIDIA TX1 GPU, FFT-Conv is 1.9× faster, 2.2× more energy-efficient, and achieves 5.6× higher throughput per layer than Direct-Conv. PENC is 10916× and 1.8× faster and 5053× and 4.3× more energy-efficient and achieves 7.5× and 1.2× higher throughput per layer than ARM A53 CPU and TX1 GPU, respectively.

ACKNOWLEDGMENT

The authors would like to thank A. Page, A. Kulkarni, and M. Hosseini for their help to improve this paper. H. Homayoun is with the Electrical and Computer Engineering Department, George Mason University, Fairfax, VA 22030 USA.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[2] J. Lu, S. Young, I. Arel, and J. Holleman, "A 1 TOPS/W analog deep machine-learning engine with floating-gate storage in 0.13 μm CMOS," *IEEE J. Solid-State Circuits*, vol. 50, no. 1, pp. 270–281, Jan. 2015.

[3] P. Vepakomma, D. De, S. K. Das, and S. Bhansali, "A-Wristocracy: Deep learning on wrist-worn sensing for recognition of user complex activities," in *Proc. IEEE 12th Int. Conf. Wearable Implantable Body Sensor Netw. (BSN)*, Jun. 2015, pp. 1–6.

[4] C. Shea, A. Page, and T. Mohsenin, "SCALENet: A scalable low power accelerator for real-time embedded deep neural networks," in *Proc. ACM 28th Ed. Great Lakes Symp. VLSI (GLSVLSI)*, 2018.

[5] A. Jafari, M. Hosseini, C. P. A. Kulkarni, and T. Mohsenin, "Binmac: Binarized neural network manycore accelerato," in *Proc. ACM 28th Ed. Great Lakes Symp. VLSI (GLSVLSI)*, 2018.

[6] S. W. Park *et al.*, "An energy-efficient and scalable deep learning/inference processor with tetra-parallel MIMD architecture for big data applications," *IEEE Trans. Biomed. Circuits Syst.*, vol. 9, no. 6, pp. 838–848, Dec. 2016.

[7] F. Ortega-Zamorano, J. M. Jerez, and L. Franco, "FPGA implementation of the C-Mantec neural network constructive algorithm," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1154–1161, May 2014.

[8] K. Simonyan and A. Zisserman. (Sep. 2014). "Very deep convolutional networks for large-scale image recognition." [Online]. Available: https://arxiv.org/abs/1409.1556

[9] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 160–167.

[10] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *Handbook Brain Theory Neural Netw.*, vol. 3361, no. 10, p. 1995, 1995.

[11] C. Szegedy *et al.*, "Going deeper with convolutions," *CoRR*, Sep. 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, Dec. 2015.

[13] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. S. Torr, "Fully-convolutional siamese networks for object tracking," *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1606.09549

[14] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 1, pp. 23–38, Jan. 1998.

[15] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. (2013). "Overseat: Integrated recognition, localization and detection using convolutional networks." [Online]. Available: https://arxiv.org/abs/1312.6229

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[17] H. Wu and X. Gu, "Towards dropout training for convolutional neural networks," *Neural Netw.*, vol. 71, pp. 1–10, Nov. 2015.

[18] S. Ioffe and C. Szegedy. (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift." [Online]. Available: https://arxiv.org/abs/1502.03167

[19] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," *CoRR*, May 2016.

[20] B. D. Rouhani, A. Mirhoseini, and F. Koushanfar, "TinyDL: Just-in-time deep learning solution for constrained embedded systems," in *Proc. Int. Symp. Circuits Syst. (ISCAS)*, 2017, pp. 1–4.

[21] N. Attaran, A. Puranik, J. Brooks, and T. Mohsenin, "Embedded low-power processor for personalized stress detection," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, to be published.

[22] A. Page, A. Jafari, C. Shea, and T. Mohsenin, "SparcNet: A hardware accelerator for efficient deployment of sparse convolutional networks," *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, p. 31, May 2017.

[23] Hitachi. (2015). *The Internet on Wheels and Hitachi, Ltd*. [Online]. Available: https://www.hitachivantara.com/en-us/pdf/white-paper/hitachi-white-pape%r-internet-on-wheels.pdf

[24] Intel. (2016). *Data is the New Oil in the Future of Automated Driving*. [Online]. Available: https://newsroom.intel.com/editorials/krzanich-the-future-of-automated-%driving/

[25] M. Malik, S. Rafatirah, A. Sasan, and H. Homayoun, "System and architecture level characterization of big data applications on big and little core server architectures," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Oct. 2015, pp. 85–94.

[26] M. Malik, K. Neshatpour, T. Mohsenin, A. Sasan, and H. Homayoun, "Big vs little core for energy-efficient hadoop computing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1480–1485.

[27] A. Jafari, A. Page, C. Sagedy, E. Smith, and T. Mohsenin, "A low power seizure detection processor based on direct use of compressively-sensed data and employing a deterministic random matrix," in *Proc. IEEE Biomed. Circuits Syst. (Biocas) Conf.*, Oct. 2015, pp. 1–4.

[28] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," *CoRR*, May 2016. [Online]. Available: http://arxiv.org/abs/1605.07678

[29] M. Wang, T. Xiao, J. Li, J. Zhang, C. Hong, and Z. Zhang, "Minerva: A scalable and highly efficient training platform for deep learning," in *Proc. NIPS Workshop, Distrib. Mach. Learn. Matrix Comput.*, 2014.

[30] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[31] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through FFTs," *CoRR*, Mar. 2013. [Online]. Available: http://arxiv.org/abs/1312.5851

[32] T. Highlander and A. Rodriguez, "Very efficient training of convolutional neural networks using fast fourier transform and overlap-and-add," *CoRR*, Jan. 2016. [Online]. Available: http://arxiv.org/abs/1601.06815

[33] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun. (2014). "Fast convolutional nets with FBFFT: A GPU performance evaluation." [Online]. Available: https://arxiv.org/abs/1412.7580?context=cs

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[35] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, Apr. 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[36] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, Dec. 2015. [Online]. Available: http://arxiv.org/abs/1512.00567
[37] A. Page, N. Attaran, C. Shea, H. Homayoun, and T. Mohsenin, "Low-power manycore accelerator for personalized biomedical applications," in *Proc. 26th Ed. Great Lakes Symp. VLSI (GLSVLSI)*, New York, NY, USA: ACM, 2016, pp. 63–68. [Online]. Available: http://doi.acm.org/10.1145/2902961.2902986
[38] A. Kulkarni, T. Abtahi, E. Smith, and T. Mohsenin, "Low energy sketching engines on many-core platform for big data acceleration," in *Proc. 26th Ed. Great Lakes Symp. VLSI (GLSVLSI)*, New York, NY, USA: ACM, 2016, pp. 57–62. [Online]. Available: http://doi.acm.org/10.1145/2902961.2902984
[39] J. Bisasky, D. Chandler, and T. Mohsenin, "A many-core platform implemented for multi-channel seizure detection," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2012, pp. 564–567.
[40] J. Bisasky *et al.*, "A 64-core platform for biomedical signal processing," in *Proc. Qual. Electron. Design (ISQED)*, Mar. 2013, pp. 368–372.
[41] M. A. G. Silva. (2013). *Modified Calibrator*. [Online]. Available: https://github.com/magsilva/calibrator
[42] J. Eaton, D. Bateman, S. Hauberg, and W. Rik, *Gnu Octave Version 3.0.1 Manual: A High-Level Interactive Language For Numerical Computations*. CreateSpace Independent Publishing Platform, 2014. [Online]. Available: http://www.gnu.org/software/octave/doc/interpreter
[43] Y. Ma, M. Kim, Y. Cao, S. Vrudhula, and J.-S. Seo, "End-to-end scalable fpga accelerator for deep residual networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.
[44] J. Shen, N. Vesdapunt, V. Boddeti, and K. M. Kitani. (2016). "In teacher we trust: Learning compressed models for pedestrian detection." [Online]. Available: https://arxiv.org/abs/1612.00478

Tahmid Abtahi received the M.S. degree in computer engineering from the University of Maryland at Baltimore County (UMBC), Baltimore, MD, USA.

He is currently a Data Scientist at Senseonics Inc., Germantown, MD, USA, where he is involved in machine-learning-based glycemic pattern recognition in diabetic patients. His current research interests include accelerating convolutional neural networks with fast Fourier transform-based algorithm optimization.

Mr. Abtahi has received the JCET Graduate Student Fellowship in 2017 for his research on embedded air quality monitoring in collaboration with the Atmospheric Physics Department, UMBC, and the National Aeronautics and Space Administration. He also received the Best Paper (Honorable Mention) Award at the 50th International Symposium on Circuits and Systems in 2017 for his work on accelerating convolutional neural network with fast Fourier transform on tiny cores.

Colin Shea received the B.S. degree in electrical engineering from Bradley University, Peoria, IL, USA, in 2003, and the M.S. degree in computer engineering from George Washington University, Washington, DC, USA, in 2008. He is currently working toward the Ph.D. degree in computer engineering at the University of Maryland at Baltimore County, Baltimore, MD, USA.

His current research interests include low-power hardware design for machine learning implementations, high-performance computing using GPUs and FPGAs, low-power design methodologies, and parallel computing.

Amey Kulkarni received the M.Tech. degree in VLSI design from the Vellore Institute of Technology, Vellore, India, in 2010, and the Ph.D. degree in computer engineering from the University of Maryland at Baltimore County (UMBC), Baltimore, MD, USA.

He was a VLSI Engineer at Silicon Interfaces and 3D Microsystems Private Limited, Navi Mumbai, India, for two years. He is currently an FPGA Engineer with DSP and computer vision focus at research and development laboratories of Velodyne LiDAR, Inc., San Jose, CA, USA. He is a Key Designer of the 192-core many-core chip called power-efficient nanocluster. He is actively researching strategies to efficiently perform computer vision and signal processing algorithms acceleration using hardware platforms, particularly FPGA and multiprocessor system-on-chip. During his academic career, he has published over 15 papers in peer-reviewed conferences and journals. His current research interests include designing real-time and low-power hardware architecture that are resource-efficient, cognitive, and trustworthy.

Dr. Kulkarni received the Best Paper (Honorable Mention) Award at the 50th International Symposium on Circuits and Systems in 2017 for his work on accelerating convolutional neural network with fast Fourier transform on tiny cores.

Tinoosh Mohsenin received the M.S. degree in electrical and computer engineering from Rice University, Houston, TX, USA, in 2004, and the Ph.D. degree in electrical and computer engineering from the University of California at Davis, Davis, CA, USA, in 2010.

She is currently an Assistant Professor at the Department of Computer Science and Electrical Engineering, University of Maryland at Baltimore County, Baltimore, MD, USA, where she directs the Energy Efficient High Performance Computing Laboratory. She has over 80 peer-reviewed journal and conference publications. She currently leads eight research projects in her laboratory which are all funded by the National Science Foundation, the Army Research Laboratory, Northrop Grumman, Boeing, Nvidia, and Xilinx. Her current research interests include designing highly accurate and energy-efficient embedded processors for machine learning, signal processing and knowledge extraction techniques for autonomous systems, wearable smart health monitoring, and embedded big data computing.

Dr. Mohsenin has served as a Technical Program Committee Member of the IEEE International Solid-State Circuits Conference Student Research, IEEE BIOMEDICAL CIRCUITS AND SYSTEMS, IEEE the International Symposium on Circuits and Systems (ISCAS), the ACM Great Lakes Symposium on VLSI (GLSVLSI), and the IEEE International Symposium on Quality Electronic Design Conferences. She was a recipient of the NSF CAREER Award in 2017, the best paper award at the GLSVLSI Conference in 2016, and the Best Paper Honorable Award at ISCAS in 2017 for developing domain-specific accelerators for biomedical, deep learning, and cognitive computing. She was the local arrangement Co-Chair for the 50th IEEE ISCAS in Baltimore. She has served as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—Part I and the IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS. She also serves as a Secretary of IEEE P1890 on Error Correction Coding for Non-Volatile Memories.