

APPROVAL SHEET

Title of Thesis: Accelerating Convolutional Neural Network with FFT on Embedded Hardware

Name of Candidate: Tahmid Syed Abtahi
Master of Science, 2017

Thesis and Abstract Approved: _____



Tinoosh Mohsenin
Assistant Professor
Department of Computer Science and
Electrical Engineering

Date Approved: 07, 25, 2017

ABSTRACT

Title of Thesis: **Accelerating Convolutional Neural Network
with FFT on Embedded Hardware**

Name of Student **Tahmid Syed Abtahi**
Computer Engineering, 2017

Thesis directed by: **Professor Tinoosh Mohsenin**
Department of Computer Science &
Electrical Engineering

Fueled by ILSVRC and COCO competitions, Convolutional Neural Network (CNN) has become important in computer vision, and natural language processing. However, state-of-the-art CNNs are computationally and memory intensive, thus energy efficient implementation on embedded platform is challenging. Recently VGGNet and ResNet showed that deep neural networks with more convolution layers and few fully connected layer can achieve lower error rates, thus reducing the complexity of convolution layers is of utmost importance. Recent trends in neural network architectures show that, fully connected layers are reduced and more emphasis is given to convolution layers. 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 for two case studies of object detection and atmospheric big data compression. For the case study of object detection, we implemented these three techniques for ResNet-20 with the CIFAR-10 dataset on a low power

domain specific many-core called Power Efficient Nano Clusters (PENC), NVIDIA Jetson TX1 GPU, and ARM Cortex A53 CPU to explore the trade off 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. Using built-in FFT instruction in PENC, the FFT-OVA-Conv performs $2.9\times$ and $1.65\times$ faster and achieves $6.7\times$ and $2.3\times$ higher throughput per watt than Direct-Conv and FFT-Conv. In ARM A53 CPU, FFT-OVA-Conv achieves $3.36\times$ and $1.38\times$ improvement in execution time and $2.72\times$ and $1.32\times$ higher throughput than Direct-Conv and FFT-Conv. In TX1 GPU FFT based convolution is $1.9\times$ faster, $2.2\times$ more energy efficient and achieves $5.6\times$ higher throughput per layer than Direct-Conv. PENC is $10,916\times$ and $1.8\times$ faster and $9,200\times$ and $7.9\times$ more energy efficient and achieves $7.5\times$ and $1.2\times$ higher throughput per layer than ARM A53 CPU and TX1 GPU, respectively. For the case study of atmospheric big data compression, we apply the proposed FFT based convolution techniques for the compression and decompression (CoDec) of LIDAR Backscattering profile from Vaisala CL31 ceilometer. We evaluate Direct-Conv, FFT-Conv, and FFT-OVA-Conv based discrete wavelet transform for compression and decompression in ARM Cortex A53 CPU. With 1 level of compression on 24hr of data at sensor and decompressing at base we achieved run time of 32.13s with throughput of 138.6 Ksample/s with 75% reduction in transmission and storage consumption. FFT-OVA-Conv based CoDec achieved $10.6\times$ and $4\times$ faster execution time than Direct-Conv and FFT-Conv based CoDec method in ARM A53 CPU.

**Accelerating Convolutional Neural Network
with FFT on Embedded Hardware**

by

Tahmid Syed Abtahi

MS Thesis, 2017

Advisory Committee:

Professor Tinoosh Mohsenin, Chair/Advisor

Professor Belay Demoz

Professor Gary M. Carter

Professor Ruben Delgado

To my parents with everlasting gratitude

ACKNOWLEDGMENTS

I owe gratitude to all those who have made this thesis possible. I would like to express my sincere gratitude to my advisor, Dr. Tinoosh Mohsenin. Her enthusiasm and remarkable foresight in this research field inspire me all the time. I greatly appreciate the countless valuable advice she gave to me for research, presentation, and communication skill. I would also like to thank my committee members, Dr. Belay Demoz, Dr. Gary M. Cater, and Dr. Ruben Delgado for all their feedbacks and guidance. I am deeply grateful to the Joint Center for Earth Systems Technology (JCET) for awarding me Graduate Student Fellowship 2016-17 which boosted my research and introduced me to the fascinating research areas of atmospheric physics. I would like to thank Dr. Amey kulkarni who has been a great mentor, true friend and inspiration in my research. I would also like thank my lab-mates Adwaya Kulkarni, Colin Shea, and Dr. Adam Page for their contributions. Finally, I would like to thank all my family members and friends for their support. Without their encouragement, I would not have been able to accomplish this.

TABLE OF CONTENTS

| | |
|---|------------|
| DEDICATION | ii |
| ACKNOWLEDGMENTS | iii |
| LIST OF FIGURES | vi |
| Chapter 1 INTRODUCTION | 1 |
| 1.1 Motivation and Problem Statement | 1 |
| 1.2 Contribution | 5 |
| 1.3 Organization of Thesis | 6 |
| Chapter 2 BACKGROUND | 8 |
| 2.1 Convolution Neural Network Layer | 8 |
| 2.1.1 Fully Connected Layer | 8 |
| 2.1.2 Convolution Layer | 9 |
| 2.1.3 Max Pooling Layer | 10 |
| 2.2 Traditional and FFT Based Convolution Techniques | 11 |
| 2.2.1 Direct Convolution (Direct-Conv) | 11 |
| 2.2.2 FFT Convolution (FFT-Conv) | 11 |
| 2.2.3 OverLap and Add FFT Convolution (FFT-OVA-Conv) | 13 |
| 2.3 Computational Complexity and Memory Access Analysis | 15 |
| Chapter 3 EMBEDDED PLATFORMS | 18 |
| 3.1 Embedded Platforms and Experimental Setup | 18 |
| 3.1.1 PENC Many-core Overview and Key Features | 18 |
| 3.1.2 ARM Cortex-A53 CPU | 24 |
| 3.1.3 NVIDIA Jetson TX1 GPU | 25 |
| Chapter 4 CASE STUDY- OBJECT DETECTION | 27 |
| 4.1 Motivation | 27 |

| | | |
|-------------------|--|-----------|
| 4.2 | Choice of Object Detection Network for Embedded Deployment | 27 |
| 4.3 | Dataset | 29 |
| 4.4 | Implementation Results | 30 |
| 4.4.1 | PENC Many-Core implementation | 30 |
| 4.4.2 | ARM Cortex-A53 CPU Implementation | 33 |
| 4.4.3 | NVIDIA TX1 Jetson GPU Implementation | 34 |
| 4.4.4 | Cross Platform Analysis | 35 |
| 4.4.5 | Comparison with Others | 36 |
| Chapter 5 | CASE STUDY- ATMOSPHERIC BIG DATA COMPRESSION | 38 |
| 5.1 | Motivation | 38 |
| 5.2 | Background | 39 |
| 5.2.1 | Mixing Layer | 39 |
| 5.2.2 | Automated Surface Observing System | 40 |
| 5.3 | Framework and Algorithms | 42 |
| 5.3.1 | Framework | 42 |
| 5.3.2 | Discrete Wavelet Transform (DWT) | 43 |
| 5.3.3 | Inverse Discrete Wavelet Transform (IDWT) | 46 |
| 5.3.4 | FFT Based DWT and IDWT | 48 |
| 5.3.5 | Local Processing | 48 |
| 5.4 | Dataset | 49 |
| 5.5 | Implementation Results | 50 |
| 5.5.1 | Simulation Results | 50 |
| 5.5.2 | Hardware Results | 50 |
| Chapter 6 | CONCLUSION | 54 |
| 6.1 | Results Summary | 54 |
| 6.2 | Future Work | 55 |
| REFERENCES | | 57 |

LIST OF FIGURES

| | | |
|-----|---|----|
| 1.1 | 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 rectified linear unit (relu), pooling (pool), fully connected (fc) only require 10% of the overall computation | 2 |
| 2.1 | A side-by-side comparison between a fully-connected and 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 fully-connected there exists a dense connectivity between inputs and neurons. | 9 |
| 2.2 | Different convolution schemes (a) Direct-Conv with computational complexity of $O(N^2K^2)$, (b) FFT-Conv with computational complexity of $O(N^2\log(N))$ (c) FFT-OVA-Conv with $O(N^2\log(K))$ computational complexity where data dimension is $N \times N$ and filter dimension is $K \times K$ | 12 |
| 2.3 | Computational complexity and memory analysis between three methods of CNN convolution for ResNet-20 and AlexNet. ResNet-20 has deeper network of 19 convolution layers and AlexNet have relatively shallow network of 5 convolution layers. C1_x, C2_x and so forth represents groups of 6 similar convolution layers in ResNet-20. FFT-OVA-Conv reduces computation $7\times$ and $10\times$ than Direct-Conv and $2.8\times$ and $2.5\times$ than FFT-Conv. FFT convolutions are on average $3\times$ better than Direct-Conv. Intermediate memory storage for FFT-Conv requires $2\times$ more space than Direct-Conv and FFT-OVA-Conv | 15 |
| 2.4 | ResNet-20 with CIFAR-10 data-set architecture, where C: convolutional layer with first row as filter size and second row as dimensions of output, ReLU: Rectified Linear Unit, pl: Global average pooling layer with first row as window size and second as output size, fc: fully connected layer | 16 |
| 3.1 | (A) Power Efficient Nano Clusters (PENC), Many-core Architecture (B) Bus-based Cluster Architecture (C) Post-layout view of bus-based cluster implemented in 65nm, 1V TSMC CMOS technology (D) Block Diagram of core architecture with FFT block (E) Post Layout implementation results of optimized bus-based cluster (consisting of 3 cores + bus + cluster Memory) (F) Heterogeneous deployment of PENC with Intel Atom CPU (Host) where pretrained model and parameters (filter weights) are used for network inference | 20 |
| 3.2 | Post layout implementation breakdown analysis of PENC many-more comprising of 192 processing cores, cluster bus, shared memory and router (A) Area Breakdown (B) Power Breakdown | 21 |

| | | |
|-----|---|----|
| 3.3 | PENC many-core simulator and compiler flow for applications written in Assembly. The simulator uses the PENC VLSI hardware statistics from Cadence post-layout results. | 22 |
| 3.4 | (A) ARM Cortex A53 CPU residing in Raspberry Pi 3B platform with power measurement setup, (B) NVIDIA Jetson TX1 GPU Development platform with integrated power measurements | 25 |
| 4.1 | Example Object detection task from ILSVRC 2014 | 28 |
| 4.2 | Sample images from the CIFAR dataset containing 32x32 color images of airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. | 29 |
| 4.3 | Data flow between cores with host streaming input feature maps to cores with different filter weights saved in cluster memory (CM) in PENC | 31 |
| 4.4 | (A) Layer wise run time breakdown for most-parallel implementation (B) Total Power (C) Layer wise throughput per Watt for most-parallel implementation (D) Energy Delay Product (EDP) for three different implementation: semi-serial implementation with 15 cores/layer, semi-parallel implementation with 22 cores/layer and most-parallel implementation with 37 cores/layer | 31 |
| 4.5 | Layer-wise 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 | 33 |
| 4.6 | Cross platform run time comparison between convolution methods in ARM A53 CPU, PENC many-core, and NVIDIA TX1 GPU | 35 |
| 4.7 | Cross platform energy efficiency and throughput per layer analysis between ARM Cortex-A53 CPU, PENC many-core, and NVIDIA TX1 GPU | 36 |
| 5.1 | Mixing layer above earth's surface and within Troposphere region | 40 |
| 5.2 | ASOS unit consisting of different sensors that measure wind speed, temperature, precipitation, cloud height etc. The Vipper SC+ radio transmitter included in the ASOS site, has limited bandwidth and thus takes long time to send data to base. The Ceilometer is used only to gather cloud base heights and the entire backscatter profile data is discarded due to lack of data transmission bandwidth capacity | 41 |
| 5.3 | Three different method for ceilometer data transfer (A) Transferring raw data over existing network, (B) Locally processing data and transfer key values, (C) Compressing data and transfer to air quality management agency (base) for reconstruction and processing. Here Compression is done with DWT and decompression is performed by IDWT operation | 42 |
| 5.4 | compression using discrete wavelet transform. HP: High Pass, LP: Low Pass, HH: High pass-High pass, HL: High pass-Low Pass, LH: Low pass-High pass, LL: Low pass-Low pass | 44 |
| 5.5 | (a) Original image (b) Four elements after decompression consisting of approximate image, horizontal details, vertical details, and diagonal details | 45 |
| 5.6 | Multilevel compression with DWT, where after each DWT operation, LL band is forwarded to the next DWT operation | 45 |

| | | |
|------|--|----|
| 5.7 | decompression using inverse discrete wavelet transform | 47 |
| 5.8 | Proposed prototypes for local processing algorithm | 49 |
| 5.9 | Matlab simulation for MLH retrieval with proposed local processing algorithm with Radiosonde launch results | 51 |
| 5.10 | MLH retrieval on original profile and multilevel reconstructed profile from 75%, 93.8%, 98.4%, 99.6% and 99.9% percent of compression | 51 |
| 5.11 | Mean squared error (MSE) and Peak Signal to Noise Ration (MSE) for reconstruc- tion | 52 |
| 5.12 | Difference between MLH retrievals from original profile and multilevel recon- structed profile from 75%, 93.8%, 98.4%, 99.6% and 99.9% percent of compression | 52 |

Chapter 1

INTRODUCTION

1.1 Motivation and Problem Statement

Neural networks have regained the momentum with tremendous error rate reduction shown by AlexNet [Krizhevsky, Sutskever, & Hinton2012a] in 2012 as compared to LeNet [LeCun *et al.*1998] architecture. Neural networks do not require extensive data pre-processing, feature extraction, or domain knowledge [Lu *et al.*2015, Vepakomma *et al.*2015, Park *et al.*2016, Ortega-Zamorano, Jerez, & Franco2014, Sohn, Shang, & Lee2014, Ngiam *et al.*2011, Balaji *et al.*2012], thus instantly became famous in computer vision [Simonyan & Zisserman2014a], natural language processing [Collobert & Weston2008], and speech recognition applications [LeCun & Bengio1995]. Over the past 4 years, an enormous amount of research has been conducted from two different perspectives: 1. Algorithm improvement, to reduce error rate on ILSVRC dataset [Szegedy *et al.*2014], [He *et al.*2015], target specific applications such as object detection and tracking [Bertinetto *et al.*2016], face identification [Rowley, Baluja, & Kanade1998], localization and mapping [Sermanet *et al.*2013] etc., and introducing layers to lower com-

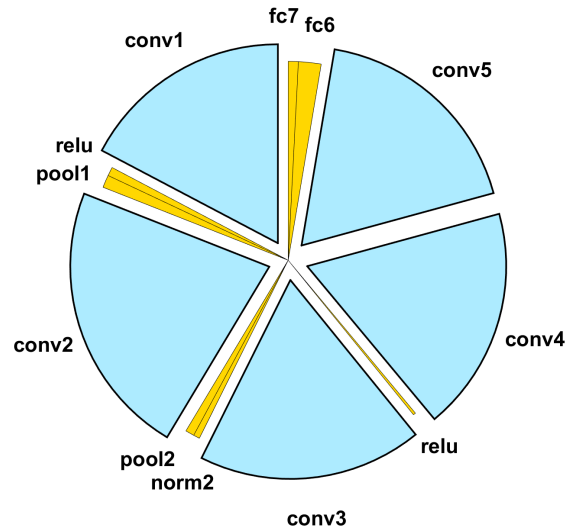


FIG. 1.1. Computation breakdown for different layers of AlexNet, a popular object detection network. Almost 90% of the computation is dominated by convolution (conv) layers. Rest of the layers such as rectified linear unit (relu), pooling (pool), fully connected (fc) only require 10% of the overall computation

putational and time complexity of training and inference [Srivastava *et al.*2014], [Wu & Gu2015], [Ioffe & Szegedy2015]. 2. Hardware improvement, to reduce memory transfers [Han *et al.*2016], [Rouhani, Mirhoseini, & Koushanfar2017], and computations of convolution layer [Page *et al.*2017], [Qiu *et al.*2016], 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 on-board including LiDARs, stereo and vision cameras, and radar, which generate significant amount of data each minute. At the same time, these applications require

accurate decision making abilities with additional rigid constraints on real-time and power consumption and very little space on board [Jafari & Mohsenin2015]. Deep convolution neural networks are an ideal candidate for such applications, however, these networks experience real-time and power consumption bottlenecks.

In convolution neural networks (CNN) two layers mainly contributed to network bottlenecks, the convolution layer is the most computationally complex whereas the fully connected layer is the most memory intensive. For example, Figure 1.1 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 CNN is extremely important.

There has been a surge in research in Deep CNN accelerators. Page et al. [Page *et al.*2017] presented SPARCNet, an FPGA-based convolutional neural network accelerator 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 $15\times$ improvement in energy efficiency [Page *et al.*2017] compared to other FPGA based accelerators while requiring less than 2W power consumption. Regean et al. [Wang *et al.*] 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 $8.1\times$ power reduction on average. Manoj et al. [Alwani *et al.*2016] proposed fused-layer CNN accelerators which focus on reducing data flow across layers and fusing multiple convolutional layer compu-

tation 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. LeCun et al. [Mathieu, Henaff, & LeCun2013] showed fast training and inference of convolutional networks through FFTs in GPU architecture. This was able to achieve significant speed up when number of feature maps is large. However, for embedded systems with limited memory resources, intermediate memory build up in FFT algorithm can pose a serious issue. Tyler et al. [Highlander & Rodriguez2016] showed CNN can be trained by overlap and add based FFT transform and potentially avoid the memory augmentation problem and attain $16.3\times$ improvement in computation time over traditional convolution implementation for a 8×8 filter and a 224×224 image. In this paper we propose FFT-Overlap and Add method to reduce computations in convolution layer.

CNNs consist of a variety of layers such as convolution, fully connected, max-pooling, batch normalization, and Rectified Linear Unit (ReLU), in which convolution and fully connected layers are called as weighted layers. Convolution layers are computationally complex due to the sliding window and enormous amounts of multiplications, whereas fully connected layers are memory intensive. FFT based convolution (FFT-Conv) is widely used to reduce computation complexity of the convolution layer [Vasilache *et al.*2014]. 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 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-Conv execution.

1.2 Contribution

The key contributions of this work are as follows:

- Detailed analysis of FFT Convolution (FFT-Conv) and FFT Overlap and Add Convolution (FFT-OVA-Conv)
- Analysis of popular CNN networks and choice of network analysis for embedded CNN deployment
- Implementation of Direct Convolution (Direct-Conv) as well as FFT-Conv and FFT-OVA-Conv techniques for ResNet-20 [He *et al.*2015] on PENC many-core, ARM A53 CPU, and NVIDIA Jetson TX1 GPU using CUDA-based 16-bit TensorFlow.
- Thorough cross platform throughput and timing performance, power and energy analysis amongst ARM Cortex A53 CPU, PENC many-core, and NVIDIA Jetson TX1 GPU for different convolution implementation of ResNet-20 on CIFAR-10 data-set

To show effectiveness of the three different convolution methods for atmospheric data

applications, we considered a case study of big data compression in Automatic Surface Observation System (ASOS) stations located at around 1000 locations all across US collecting valuable atmospheric data 24/7. The major contributions of our work on this application includes

- Introducing discrete wavelet transform based compression decompression (CoDec) scheme to solve ASOS station's data transmission bandwidth limitation problem
- Implementation of software and hardware prototypes for proposed big data CoDec scheme in ARM Cortex A53 CPU
- Implementation of FFT Convolution based compression decompression scheme for atmospheric big data application in ARM Cortex A53 CPU

1.3 Organization of Thesis

In the second chapter we review concepts, background and analyze analytic performance of different convolution methods of different networks. In third chapter we introduce different embedded platforms and their experimental setups. In fourth chapter we discuss motivation, implementation and results for the case study of object detection network ResNet 20 on PENC, Arm Cortex A53 CPU, and NVIDIA TX1 GPU. In the fifth chapter, for the case study of atmospheric big data compression, we discuss the background of ASOS transmission bandwidth limitations, concepts of CoDec and FFT based

CoDec and implementations in Arm A53 CPU. In sixth chapter summary of the results and future works are discussed.

Chapter 2

BACKGROUND

2.1 Convolution Neural Network Layer

In deep CNN architectures, there exists a large variety of layer types including fully-connected, 1-D/2-D convolutional, pooling, batch normalization, and other specialized forms. There are also activation functions, which are often non-linear such as *sigmoid*, *tanh*, and *ReLU*, that can be treated as separate layers. Out of all of the computational layers, fully-connected 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.

2.1.1 Fully Connected Layer

In Fully Connected (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 fully-connected layers is that for high-dimensional data, the dense connectivity and large parameter set

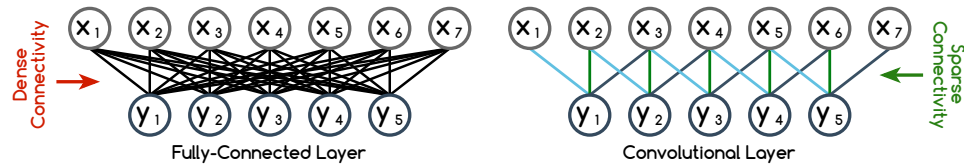


FIG. 2.1. A side-by-side comparison between a fully-connected and 1-D convolutional layer for input X and output Y . The edges designate multiplication between input and corresponding weight. For convolutional, edge color designates tied weights. In fully-connected there exists a dense connectivity between inputs and neurons.

make it difficult to learn a meaningful transformation.

2.1.2 Convolution Layer

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

Figure 2.1 shows a side-by-side comparison of a fully-connected layer and a corresponding 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 fully-connected layer requires performing approximately $2 \times 5 \times 7 = 70$ operations and storing $7 \times 5 = 35$ weights, whereas the convolutional layer requires performing approximately $2 \times 5 \times 3 = 30$ operations and storing 3 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.

2.1.3 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.

2.2 Traditional and FFT Based Convolution Techniques

2.2.1 Direct Convolution (Direct-Conv)

Direct convolution involves multiplication addition (MAC) between a data patch and corresponding filter. Patches are constructed by striding along the data.

$$(2.1) \quad d(n) \star f(n) = \sum_{m=-\infty}^{\infty} d(m-n) \times f(m) dm$$

For two dimensional data with $N \times N$ dimensions and filter of size $K \times K$ ($N > K$), the number of patches can be calculated by $(N - K + 1)/stride$. For multichannel data, MAC results between patches of data channels and filter channels are summed to obtain a pixel of the output. Figure 2.2A shows convolution between a single channel 4x4 data and a 2x2 filter. 9 patches undergo MAC operation to obtain 9 pixels of the 3x3 output.

2.2.2 FFT Convolution (FFT-Conv)

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

$$(2.2) \quad d(n) \star f(n) = \mathcal{F}^{-1} \left\{ \mathcal{F}(d(n)) \times \mathcal{F}(f(n)) \right\}$$

In this method, first both filter and data are transformed into frequency domain by FFT of $(N+K-1)$ length. Therefore filter FFT coefficients require additional intermediate

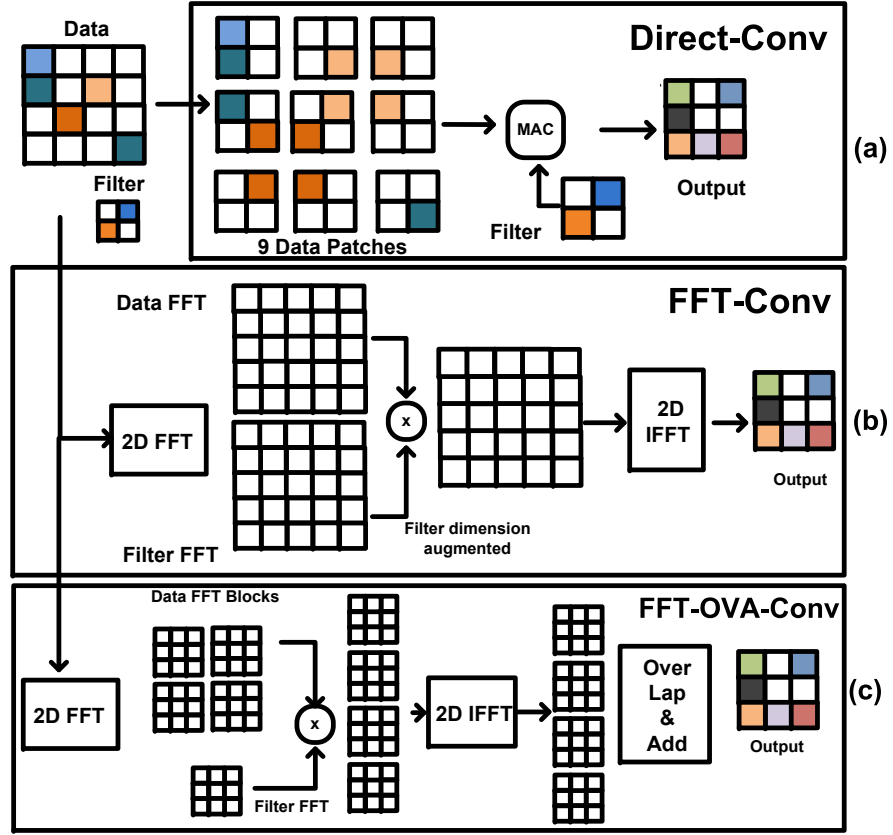


FIG. 2.2. Different convolution schemes (a) Direct-Conv with computational complexity of $O(N^2 K^2)$, (b) FFT-Conv with computational complexity of $O(N^2 \log(N))$ (c) FFT-OVA-Conv with $O(N^2 \log(K))$ computational complexity where data dimension is $N \times N$ and filter dimension is $K \times K$

memory. Then data FFT and filter FFT are element wise multiplied and fed to a Inverse FFT to obtain output as showed in algorithm 1. Figure 2.2B shows filter dimension of 2x2 is augmented to 4x4 due to initial FFT transform.

2.2.3 OverLap and Add FFT Convolution (FFT-OVA-Conv)

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 Convolution is performed on these segmented data and then block outputs are aligned and added as showed in equation 5.1 and explained in algorithm 2.

$$(2.3) \quad d(n) \star f(n) = \sum_k \left(\mathcal{F}^{-1} \left\{ \mathcal{F}(d(n - kL)) \times \mathcal{F}(f(n)) \right\} \right)$$

Figure 2.2C shows a 4x4 data to be segmented into 4 blocks of 2x2 size (3x3 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 same dimension. Outputs of FFT-Conv from 4 blocks of size 3x3 are aligned and added to obtain the 3x3 output.

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
-

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 $K \times K$ dimension
2. Zero-pad filter to the dimension of $(K+K-1)$
3. Do Fast Fourier Transform of filter to get F
4. each O_{ij} data block 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} 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}

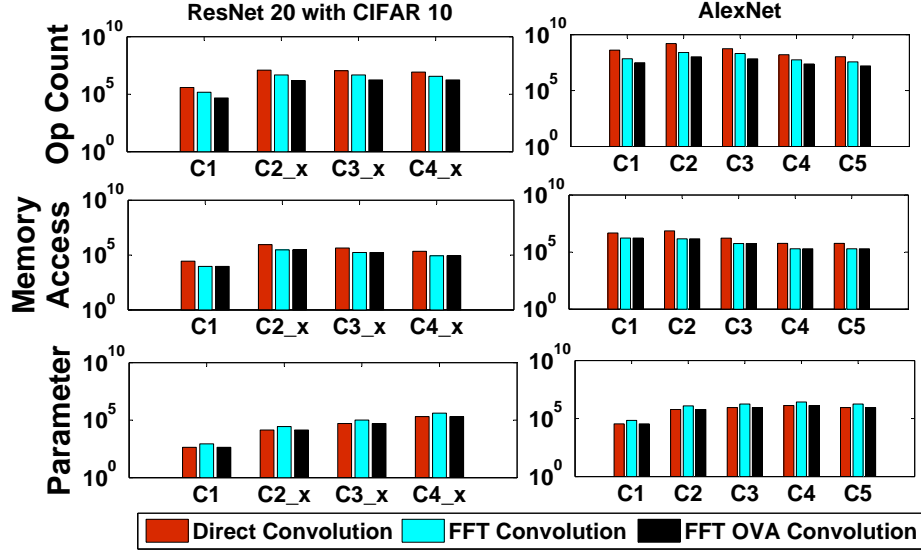


FIG. 2.3. Computational complexity and memory analysis between three methods of CNN convolution for ResNet-20 and AlexNet. ResNet-20 has deeper network of 19 convolution layers and AlexNet have relatively shallow network of 5 convolution layers. C1_x, C2_x and so forth represents groups of 6 similar convolution layers in ResNet-20. FFT-OVA-Conv reduces computation $7\times$ and $10\times$ than Direct-Conv and $2.8\times$ and $2.5\times$ than FFT-Conv. FFT convolutions are on average $3\times$ better than Direct-Conv. Intermediate memory storage for FFT-Conv requires $2\times$ more space than Direct-Conv and FFT-OVA-Conv

2.3 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 being $O(N^2 K^2)$. Depending upon stride, 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

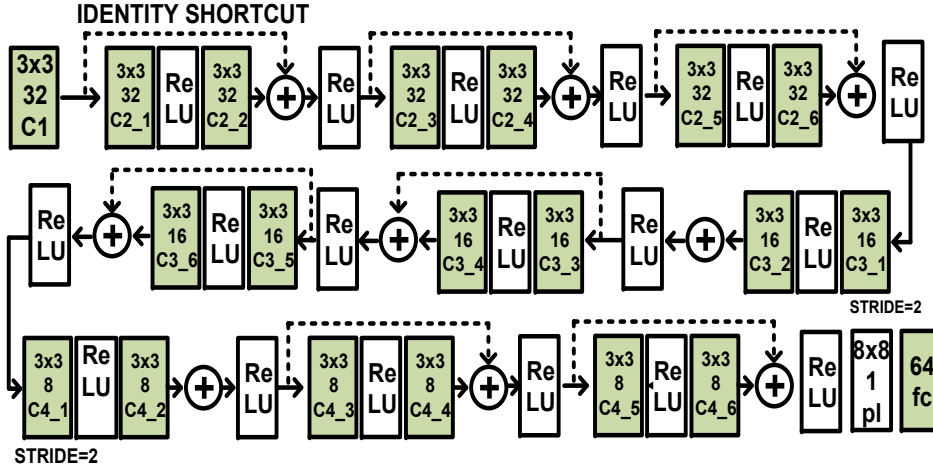


FIG. 2.4. ResNet-20 with CIFAR-10 data-set architecture, where C: convolutional layer with first row as filter size and second row as dimensions of output, ReLU: Rectified Linear Unit, pl: Global average pooling layer with first row as window size and second as output size, fc: fully connected layer

a higher point FFT and require additional space. In FFT-Conv, computational complexity is reduced to $O(N^2 \log(K))$, and there is minimal additional memory storage requirements.

We evaluate computational complexity and analyze memory requirements for ResNet-20 [He *et al.* 2015] architecture with CIFAR-10 and AlexNet [Krizhevsky, Sutskever, & Hinton 2012a]. 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 showed in Figure 2.4. AlexNet consists of 5 sequential convolutional layers with three different filter sizes of 11×11 , 5×5 , 3×3 . All filters in ResNet-20 are 3×3 .

Figure 2.3 shows that for both AlexNet and ResNet-20, FFT-OVA-Conv reduces computation $7\times$ and $10\times$ compared to Direct-Conv and $2.8\times$ and $2.5\times$ compared to FFT-Conv which is evident in initial layers where input image size is larger than filter kernel. In terms of memory access, both types of FFT convolution 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.

Chapter 3

EMBEDDED PLATFORMS

3.1 Embedded Platforms and Experimental Setup

3.1.1 PENC Many-core Overview and Key Features

PENC many-core accelerator is a homogeneous architecture that consists of in-order tiny processors with a 6 stage pipeline, a RISC-like DSP instruction set and a Harvard Architecture model [Page *et al.*2016, Kulkarni *et al.*2016]. 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 intra-cluster communication directly via a bus and inter-cluster communication through a hierarchical routing architecture. Each cluster also contains a shared memory. Figure 3.1 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 SoC Encounter and results for one cluster are summarized in Figure 3.1.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 4 processing cores without shared memory, which was ideal for DSP kernels with minimal data storage. Since CNN require 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. Below are the key characteristics of the PENC many-core platform.

3.1.1.1 Bus-based Cluster Cores use the `IN` and `OUT` instructions to communicate with each other. When a core executes an `OUT` instruction, the data and relevant addressing information is packetized and sent to its output FIFO through a bus. When data

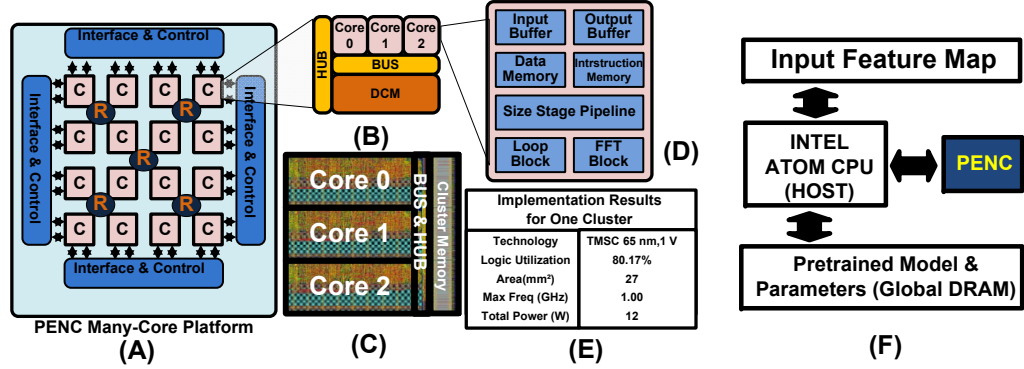


FIG. 3.1. (A) Power Efficient Nano Clusters (PENC), Many-core Architecture (B) Bus-based Cluster Architecture (C) Post-layout view of bus-based cluster implemented in 65nm, 1V TSMC CMOS technology (D) Block Diagram of core architecture with FFT block (E) Post Layout implementation results of optimized bus-based cluster (consisting of 3 cores + bus + cluster Memory) (F) Heterogeneous deployment of PENC with Intel Atom CPU (Host) where pretrained model and parameters (filter weights) are used for network inference

is present in a core's output FIFO, it requests to use the cluster bus. The bus then arbitrates between requests, only granting those whose transactions can be completed. The bus treats each transmission of data as a single transaction since it behaves with a simple push or data-driven protocol. The bus is used for intra-cluster communication. This includes a round-robin arbiter which chooses the next node to grant access based on round-robin scheme. Once the node gets access, it wraps the processing core pipeline with layers of buffering and is the main level in the PENC architecture that interacts with the bus. The destination core is used by the bus to forward the packet to the appropriate location, and the source core is used by the requesting node to satisfy its corresponding IN instruction. Based on the destination address and the data fields, the recipient core stores the address of the data.

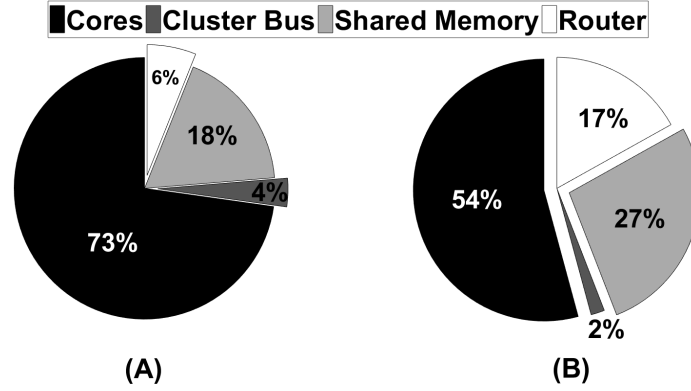


FIG. 3.2. Post layout implementation breakdown analysis of PENC many-more comprising of 192 processing cores, cluster bus, shared memory and router (A) Area Breakdown (B) Power Breakdown

3.1.1.2 Domain Specific Customization of Instruction Sets Customizing a processor's instruction set for a particular computing domain is an efficient way of improving the processors performance. Designing an application-specific hardware for each given application is expensive, hence a customized instruction set in the many-core can have a remarkable effect on power and area. The PENC architecture is optimized to best suite for machine learning kernels [Page *et al.*2016]. 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 specifying, the address in memory containing the real and imaginary values for inputs are retrieved from the hardware block. Figure 3.2 shows the post-layout implementation breakdown analysis of optimized PENC many-core comprising of 192 Processing cores, bus cluster, shared memory and router with Figure 3.2A showing the area breakdown and Figure 3.2B showing the power breakdown. These results

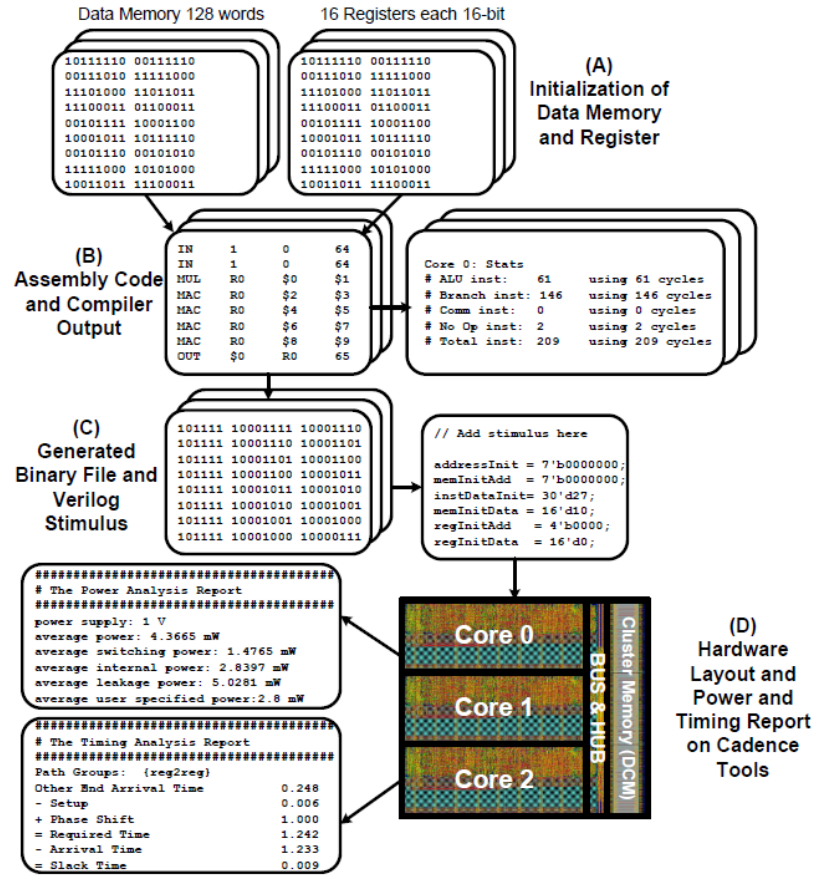


FIG. 3.3. PENC many-core simulator and compiler flow for applications written in Assembly. The simulator uses the PENC VLSI hardware statistics from Cadence postlayout results.

are obtained after Place and Route using Cadence Encounter for 65 nm technology. The area results come from the post-layout report, the power results are obtained from the Encounter power analysis with careful consideration of activity factor, capacitance, IR drop and rail analysis. These results are used to compare with the off-the-shelf processors.

3.1.1.3 Efficient Cluster Memory Access Architecture While the lightweight cores are ideal for DSP kernels that require minimal static data [Bisasky, Chandler,

& Mohsenin2012, Bisasky, Homayoun, & others2013], ML kernels often require larger amounts of memory for their model data. This is addressed with the distributed cluster-level shared memory (DCM), that is interfaced to the bus. The shared memory within a cluster consists of 3 instances of SRAM cells of memory size 1024x16 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 two memory instructions: LD and ST. The maximum depth of the cluster memory is 2^{16} words since registers and data memory are both 16-bits 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. Referencing data from the cluster memory has latency and requires a separate instruction, which reduces the overall instructions per cycle that the pipeline can complete. However, the LD and ST instructions enable the use of a much larger addressable space, which allows the PENC to support machine learning and deep learning applications.

3.1.1.4 PENC Platform Evaluation Setup For the PENC many-core, we developed stand-alone simulator and compiler that take user’s code and post-layout hardware results as seen in Figure 3.3. A 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 eas-

ier. Each task of algorithm is first implemented in assembly language on every processing core using 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 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 hardware design of the many-core platform(in verilog) and simulated using Cadence NC-Verilog as seen in Figure 3.3. 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 (ALU), branch, and communication instructions which are used for the throughput and energy analysis of the PENC many-core architecture.

3.1.2 ARM Cortex-A53 CPU

The ARM Cortex-A53 processor provides an balance between performance and power-efficiency. The Cortex-A53 is equipped with 32-bit and 64-bit instruction sets and 8-stage pipeline, efficient data fetch and access mechanism to maximize performance. It's low power and low-cost footprint makes it suitable for deployment into mobile processors, networking and automotive systems. It can be deployed as standalone processor or be paired with more powerful Cortex-A CPU such as Cortex-A57, Cortex-A72 or Cortex-A35 CPU clusters using ARM's big.LITTLE technology.

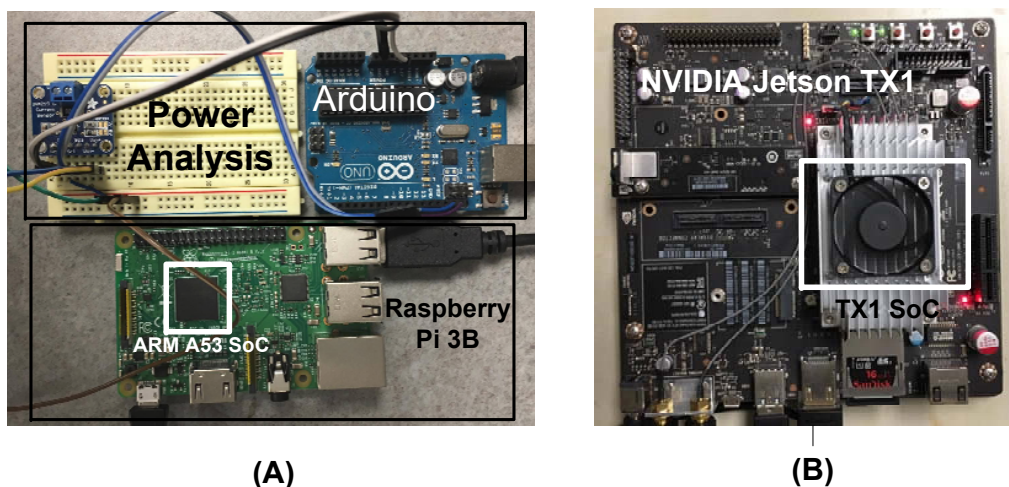


FIG. 3.4. (A) ARM Cortex A53 CPU residing in Raspberry Pi 3B platform with power measurement setup, (B) NVIDIA Jetson TX1 GPU Development platform with integrated power measurements

Cortex-A53 CPU residing on raspberry pi 3B is evaluated for energy and power consumption by collecting the current consumed at the board level as showed in Figure 3.4A. 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 Figure 3.4A. The average active current is used in calculations for the platform as is the average current when the platform is executing code.

3.1.3 NVIDIA Jetson TX1 GPU

The NVIDIA Jetson TX1, as shown in Figure 3.4B, is a System-on-Chip (SoC) combining the 256-core Maxwell graphics processing unit (GPU) and an ARM Cortex-A57 processor. The A57 processor configuration consists of four ARM Cortex-A57 proces-

sors. Each ARM A57 CPU has a 48KB L1 data and 32KB instruction cache supporting 128-bit NEON™ general-purpose SIMD instructions. All processors in the configuration have shared access to a 2MB L2 cache. Power monitoring for this platform comes from the TX1's bus addressable discrete current/power monitor. 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.

Chapter 4

CASE STUDY- OBJECT DETECTION

4.1 Motivation

Object detection is one the most popular tasks and benchmarks in computer vision research and application. Figure 4.1 shows an example object detection task from IMAGENET Large Scale Visual Recognition Challenge 2014 (ILSVRC2014) where three different objects (Person, dog, chair) in the picture are detected and labelled. The Advent of various IOT devices such as drones and driver-less cars with similar object detection capacity has stirred the need for on-board real-time embedded object detection. Therefore, to demonstrate effectiveness of FFT based convolutions over traditional convolution method we choose convolution neural networks for object detection as our first case study.

4.2 Choice of Object Detection Network for Embedded Deployment

For CNN deployment in embedded devices and tiny cores, networks that require lower number of parameters without sacrificing considerable accuracy are preferred. Table 4.1

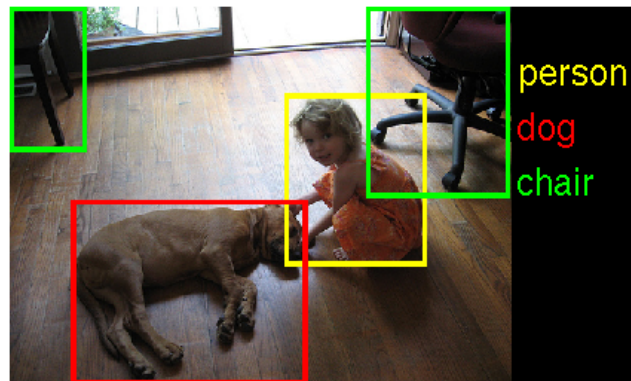


FIG. 4.1. Example Object detection task from ILSVRC 2014

shows details of popular object detection networks such as AlexNet [Krizhevsky, Sutskever, & Hinton2012b], VGG-B [Simonyan & Zisserman2014b], GoogLeNet [Szegedy *et al.*2014], VGG-D, Inception-V3 [Szegedy *et al.*2015] and ResNet20 [He *et al.*2015]. In recent years, networks have more convolution layers (CV) and only one fully connect (FC) layer at the end for classification. For example, Inception-V3 has 77 CV layers and 1 FC layers while previous predecessors such as AlexNet consists of only 5 CV layers but 3 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 Figure 2.4 consisting of 19 CV layers and 1 FC layers which requires 0.27M CV parameter is a good candidate for embedded CNN deployment. Secondly, the error rate must not be compromised while going for networks with lower CV parameters. ResNet-20 has a 8.75% error rate which is 1.7x time better than AlexNet and almost similar

Table 4.1. Parameter complexity and Accuracy of Popular Object Detection Networks
(CV = Convolution, 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 Param | 2.3M | 9.2M | 4.8M | 14.7M | 12M | 0.27M | |
| FC Param | 57.7M | 123.6M | 1M | 123.6M | 2M | 0.1K | |
| Top-5 Error | 15.30% | 9.60% | 9.20% | 7.20% | 5.60% | 8.75% | 5.10% |

to human efficiency. Therefore, in this work, ResNet-20 was considered for evaluation of CNN and FFT techniques on the embedded platforms.

4.3 Dataset

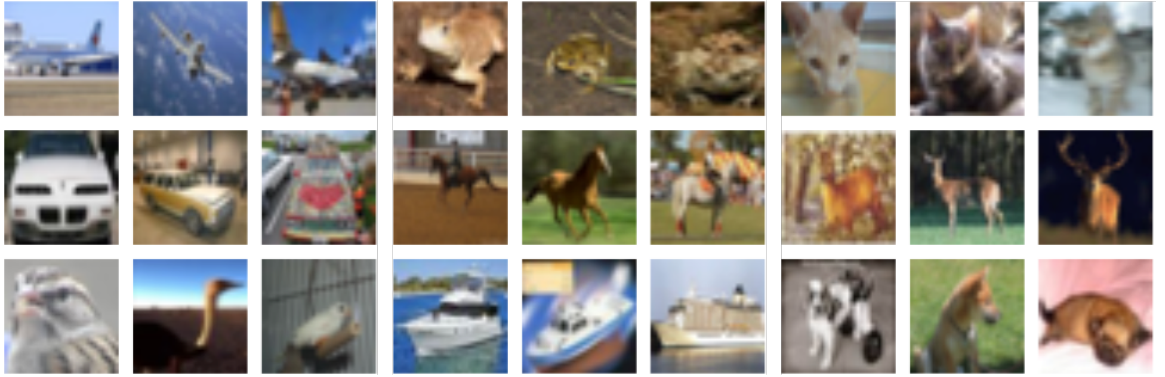


FIG. 4.2. Sample images from the CIFAR dataset containing 32x32 color images of airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

In this work we utilize the CIFAR-10 dataset with ResNet-20 network. The dataset consists of 60,000 32x32 color images with 10 classes, respectively. The image labels

include airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks [Krizhevsky & Hinton2009].

4.4 Implementation Results

4.4.1 PENC Many-Core implementation

ResNet-20 with CIFAR-10 is considered for evaluating convolution performance on PENC many-core, NVIDIA Jetson TX1, and ARM A53 CPU. Figure 3.1(F) shows the heterogeneous platform consisting of PENC as convolution accelerator and Intel Atom CPU as host performing task scheduling, data marshaling and post processing. Pre-trained model parameters are stored on global DRAM shared between host processor and PENC accelerator. 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 cluster memories as showed in Figure 4.3. Three different levels of parallelism are implemented based on storage and scope of ILP: semi-serial, semi-parallel and most-parallel. In semi-serial implementation, number of cores is assigned as per minimum number of cores required to store total layer filter weights, which is on average 15 cores per layer for ResNet-20. In most-parallel implementation, core number is assigned equal to the number

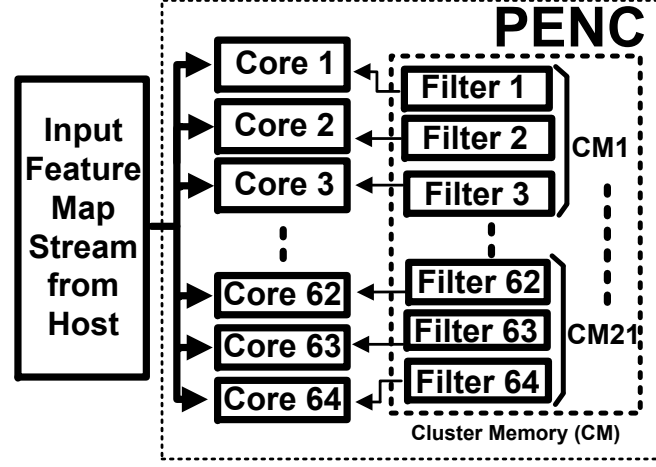


FIG. 4.3. Data flow between cores with host streaming input feature maps to cores with different filter weights saved in cluster memory (CM) in PENC

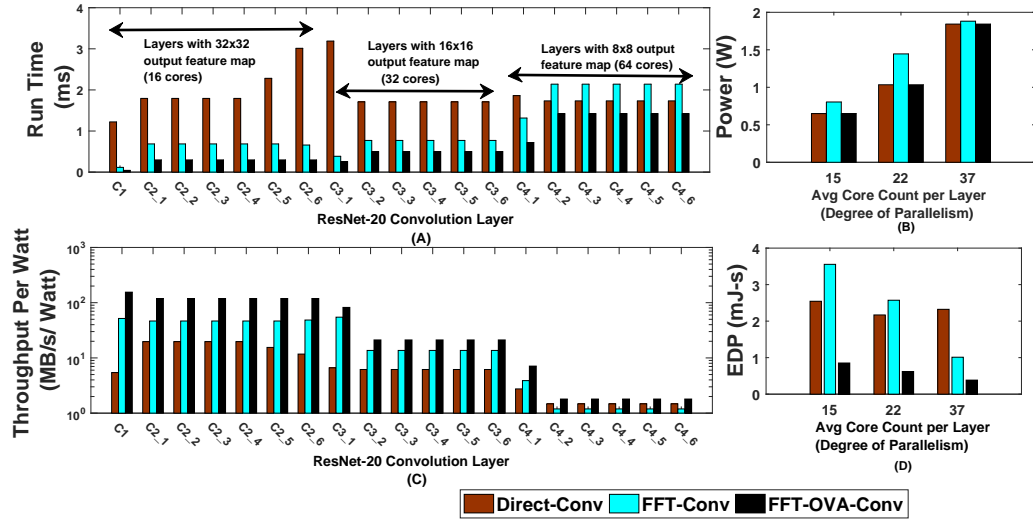


FIG. 4.4. (A) Layer wise run time breakdown for most-parallel implementation (B) Total Power (C) Layer wise throughput per Watt for most-parallel implementation (D) Energy Delay Product (EDP) for three different implementation: semi-serial implementation with 15 cores/layer, semi-parallel implementation with 22 cores/layer and most-parallel implementation with 37 cores/layer

of filters, thus achieving maximum ILP which results in 36 cores per layer on average. FFT based convolutions are implemented with PENC many-core FFT instruction. 3×3 Filters are zero padded to either 8 or 16 dimension as per size of the data. If data dimension is same or higher than 16 such as in $C1$, $C2_x$, $C3_x$ layers, 16 point FFT is used. Here, $C1$ represents first convolution layer, $C2_x$ second layer group with 32×32 data dimension and so forth. 2D FFT is performed by first finding 1D FFT of data in x direction followed by 1D FFT in y direction. Since intermediate memory is created in FFT-Conv after first FFT, we use cluster memory from a nearby idle cluster for additional storage. Therefore each cluster is surrounded by a storage cluster where cores are idle but cluster memory and corresponding routers are used. FFT-OVA used only 8-point FFT operation as data is segmented to 3×3 blocks.

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 | 38 | 5 | 2.5 |
| | FFT-Conv | 67.2 | 53 | 6 | 3.5 |
| | FFT-OVA-Conv | 31 | 28 | 18 | 0.9 |
| PENC (Most-parallel) | Direct-Conv | 36 | 65 | 10 | 2.3 |
| | FFT-Conv | 21 | 50 | 30 | 1 |
| | FFT-OVA-Conv | 12 | 31 | 60 | 0.4 |

FFT-OVA-Conv improves total run time by $2.9 \times$ and $1.65 \times$ than Direct-Conv and FFT-Conv. Figure 4.4(A) shows the layer-wise run time breakdown for most parallel im-

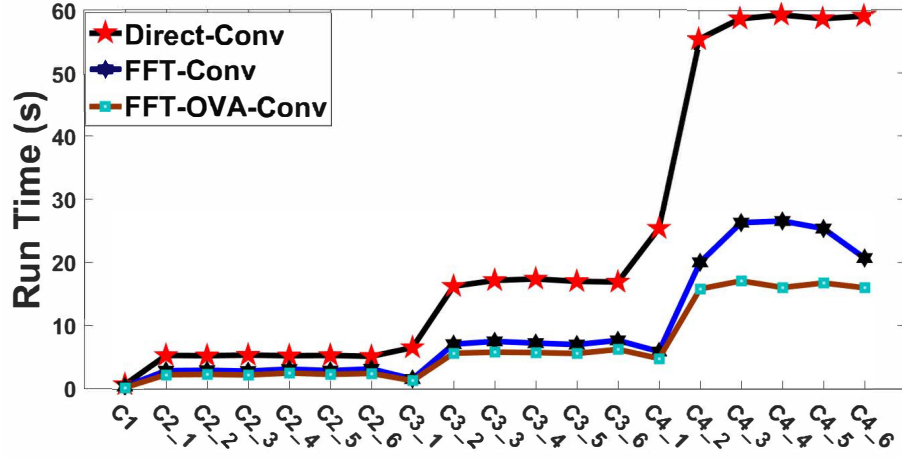


FIG. 4.5. Layer-wise run time for ARM Cortex-A53 CPU running at 1.2 GHz for serial implementation of ResNet-20 with CIFAR-10 network for different convolution methods

plementation. Since active clusters in FFT-Conv are surrounded by storage clusters for additional memory, data transmission cycles are longer and hence its run time exceeds both counterparts in final layers. Power increases in FFT-Conv by $1.4\times$ than both counterparts due to additional router and storage cluster memory power as showed in Figure 4.4(B). Throughput per watt and Energy Delay Product (EDP) improve in FFT-OVA-Conv by $6.74\times$, $2.3\times$ and $6\times$, $2.6\times$ respectively than Direct-Conv and FFT-Conv.

4.4.2 ARM Cortex-A53 CPU Implementation

We implemented ResNet-20 on low power ARM Cortex-A53 CPU running at 1.2 GHz frequency residing on Raspberry Pi 3-B using existing software libraries. GNU Octave [Eaton *et al.* 2014] high level programming language is used to run ResNet-20 model

with user defined serial functions for convolutions, ReLU and average pooling operation. Timing results are obtained from Octave’s time measurement function. Figure 4.5 shows the layer-wise timing breakdown for three different convolution methods. FFT-OVA-Conv achieves $3.36\times$ and $1.38\times$ improvement in execution time and $2.72\times$ and $1.32\times$ better throughput than Direct-Conv and FFT-Conv.

4.4.3 NVIDIA TX1 Jetson GPU Implementation

For the ResNet-20 with CIFAR-10 dataset implementation, the embedded GPU is used as an efficient accelerator. The dataset 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 dataset. 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 work we implemented the ResNet-20 using both Direct-Conv and FFT based convolution. FFT based convolution is $1.9\times$ faster, $2.2\times$ more energy efficient and achieves $5.6\times$ better throughput per layer than Direct-Conv.

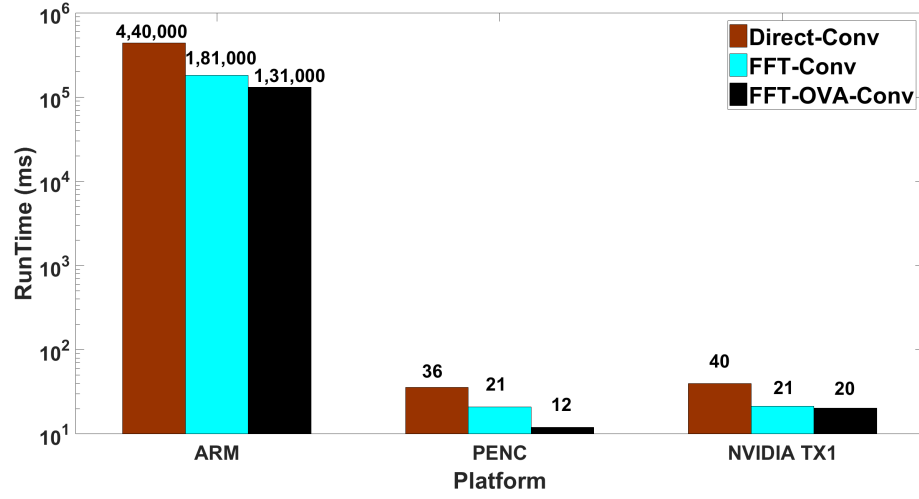


FIG. 4.6. Cross platform run time comparison between convolution methods in ARM A53 CPU, PENC many-core, and NVIDIA TX1 GPU

4.4.4 Cross Platform Analysis

In this work, the software implementation consists of general purpose embedded systems such as 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. Across three different general purpose software platforms, PENC is $10,916\times$ and $1.8\times$ faster than ARM A53 CPU and NVIDIA Jetson TX1 GPU, respectively as shown in Figure 4.6. In terms of energy, PENC is $9,290\times$ and $7.9\times$ more energy efficient than ARM A53 CPU and NVIDIA Jetson TX1 GPU. In terms of throughput per layer, PENC shows $7.5\times$ and $1.2\times$ improvement over ARM A53 CPU and NVIDIA Jetson TX1 GPU, respectively. Figure 4.7 shows energy efficiency versus throughput per layer across and

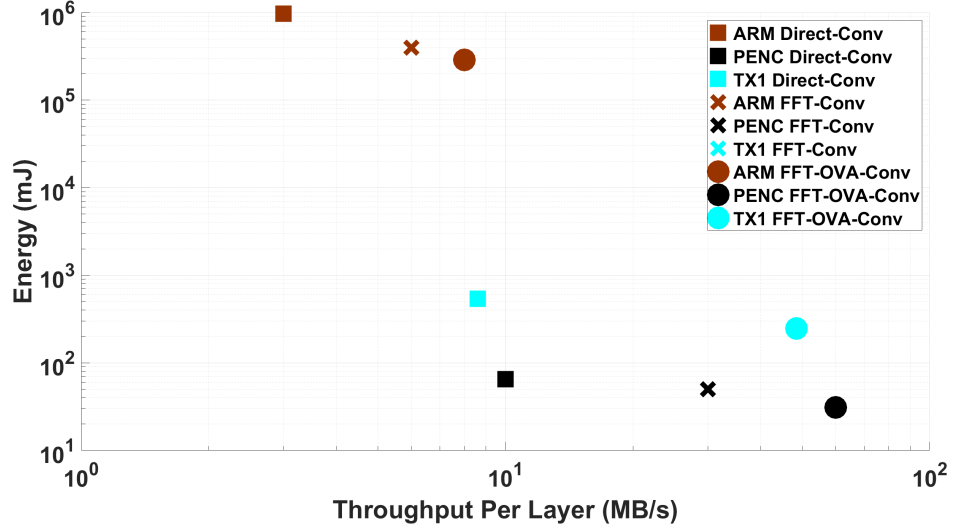


FIG. 4.7. Cross platform energy efficiency and throughput per layer analysis between ARM Cortex-A53 CPU, PENC many-core, and NVIDIA TX1 GPU

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 FFT-OVA-Conv implementation has the highest throughput with least amount of energy consumption among TX1 and ARM A53 implementations. In terms of FFT-Conv implementation between PENC and TX1 GPU, TX1 shows $1.6\times$ better throughput with expense of $4.9\times$ increased energy consumption.

4.4.5 Comparison with Others

Although there are several works on CNN implementation, we couldn't find implementation for ResNet-20 on CIFAR-10 dataset. Yufei [Ma *et al.* 2017] showed end-to-end scalable FPGA accelerator for ResNet-50 and ResNet-152 on Altera Arria-10 GX1150

FPGA with 20 nm technology and achieved 27.2 ms and 71.71 ms of execution time with 16-bit data precision and 69% DSP and 80-93% RAM utilization.

Jonathan [Shen *et al.*2016] showed hand-designed features fed to a ResNet-200 for specific task of pedestrian detection achieved 24 ms run time in Jetson TX1 GPU with 64M parameters which is $1.12\times$ better than our implementation in TX1 GPU without scaling. This arise from the fact that, during inference we restricted batch size to one image to keep similarity between implementations on rest of the platforms such as PENC, and ARM A53 CPU. Using bigger batch size will pipeline the computation and reduce overall setup time in TX1 and can achieve significant speed-up.

Chapter 5

CASE STUDY- ATMOSPHERIC BIG DATA COMPRESSION

5.1 Motivation

For the case study of atmospheric big data compression, constant collection and processing of meteorological data is a critical issue in atmospheric science. Nearly 1000 Automatic Surface Observation Systems (ASOS) located across USA, gather 24/7 atmospheric data including parameters such as temperature, visibility, precipitation, wind direction, cloud base height and LIDAR backscatter profile. Analyzing these atmospheric big data is crucial for weather forecasting, climate modelling and atmospheric research. However, all the LIDAR data is discarded due to bandwidth constraints and storage limitation. To overcome the bandwidth and storage restriction at sensor, we propose a framework for compression-decompression scheme in embedded processor that can be used in the current systems. Sensor data is compressed by Discrete Wavelet Transform (DWT) and transmitted

to the base. Received data is decompressed at the base with inverse DWT algorithm and is ready for further processing or storage. We also modified DWT and IDWT processes by replacing conventional convolution with FFT-based convolutions to minimize data flow in hardware and achieve faster execution time and throughput. Furthermore, to evaluate the performance of our framework, we proposed a less computationally intensive algorithm based on analyzing gradient distribution of LIDAR backscatter profile and is suitable for hardware deployment for Mixing Layer Height (MLH) retrieval. This processing algorithm is used to compare results obtained from original and reconstructed profile.

5.2 Background

5.2.1 Mixing Layer

In meteorology the Mixing Layer or Planetary Boundary Layer (PBL) is the lowest part of the atmosphere, adjacent to earth's surface. Its behavior is closely influenced by surface contact and is defined by exchange of heat, moisture and momentum with the surface. Mixing layer usually responds to changes in surface radioactive forcing in an hour or less. In this layer physical quantities such as flow velocity, temperature, moisture, etc., display rapid fluctuations (turbulence) and vertical mixing is strong. Above the mixing layer is the "free atmosphere" which is usually non-turbulent, or only intermittently turbulent. Figure 5.1 shows the mixing layer position between earth's surface and free atmosphere. Mixing layer height is one of the diagnostics for uncertainties in air quality monitoring

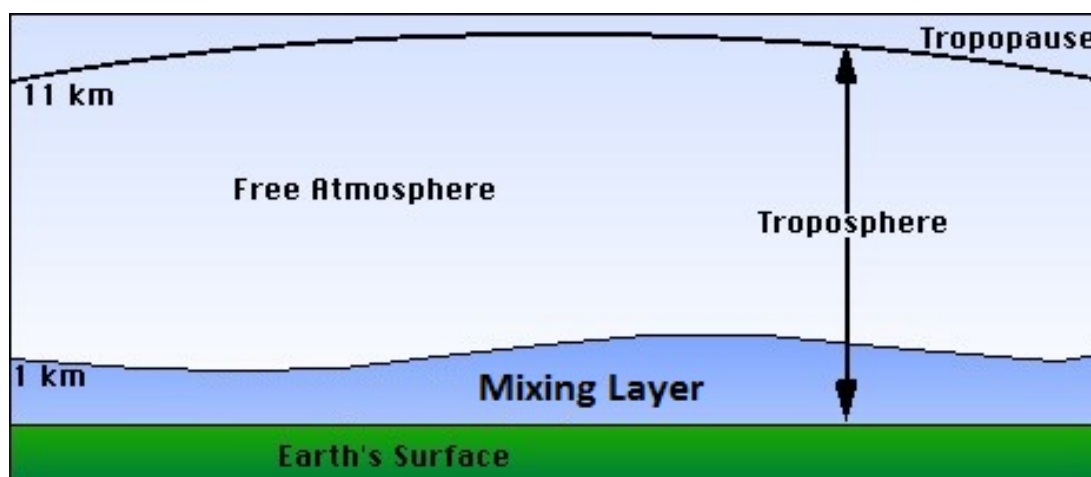


FIG. 5.1. Mixing layer above earth's surface and within Troposphere region

models [Angevine, White, & Avery1994]. Mixing layer is very crucial because it contains majority of the aerosols and its height determines the volume available for aerosol dispersion [Baars2007].

5.2.2 Automated Surface Observing System

Automated Surface Observing System (ASOS) units are automated sensor suites that are designed to serve meteorological and aviation observing needs. There are currently more than 900 ASOS sites in the United States collecting atmospheric data 24/7 including wind speed, visibility, temperature, dew point, precipitation, cloud base height, and LIDAR backscatter profile. These systems generally report at hourly intervals, but also report special observations if weather conditions change rapidly and cross aviation operation thresholds. ASOS uses Vipper SC+ radio transmitter to connect to national weather service stations and due to limited bandwidth of the transmitter, ASOS stations only transmit the

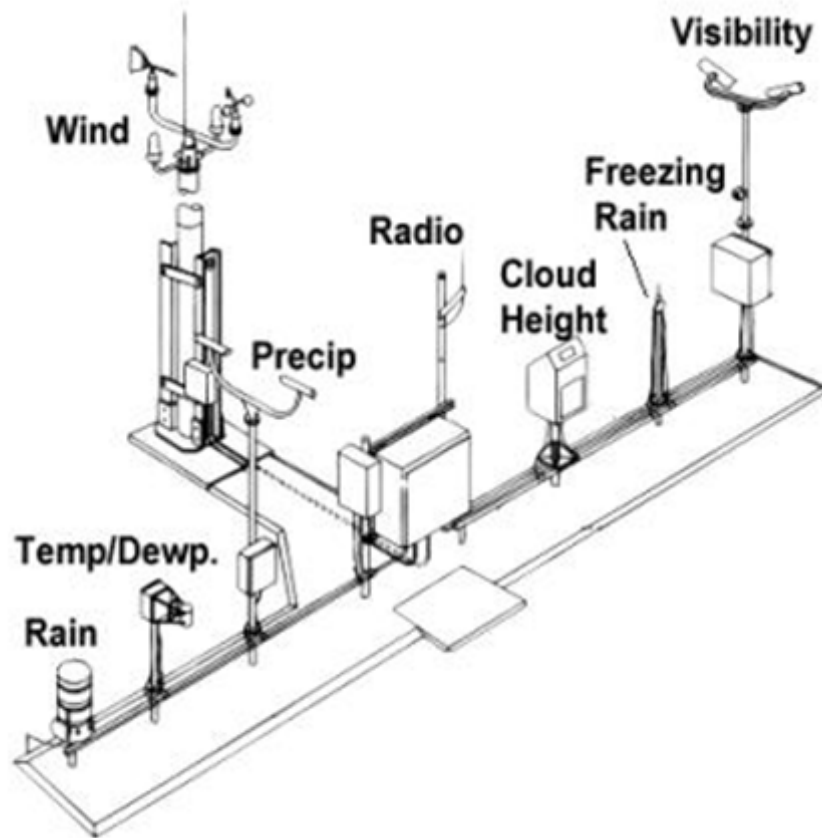


FIG. 5.2. ASOS unit consisting of different sensors that measure wind speed, temperature, precipitation, cloud height etc. The Vipper SC+ radio transmitter included in the ASOS site, has limited bandwidth and thus takes long time to send data to base. The Ceilometer is used only to gather cloud base heights and the entire backscatter profile data is discarded due to lack of data transmission bandwidth capacity

cloud height information from the ceilometer and discards the entire backscatter profile.

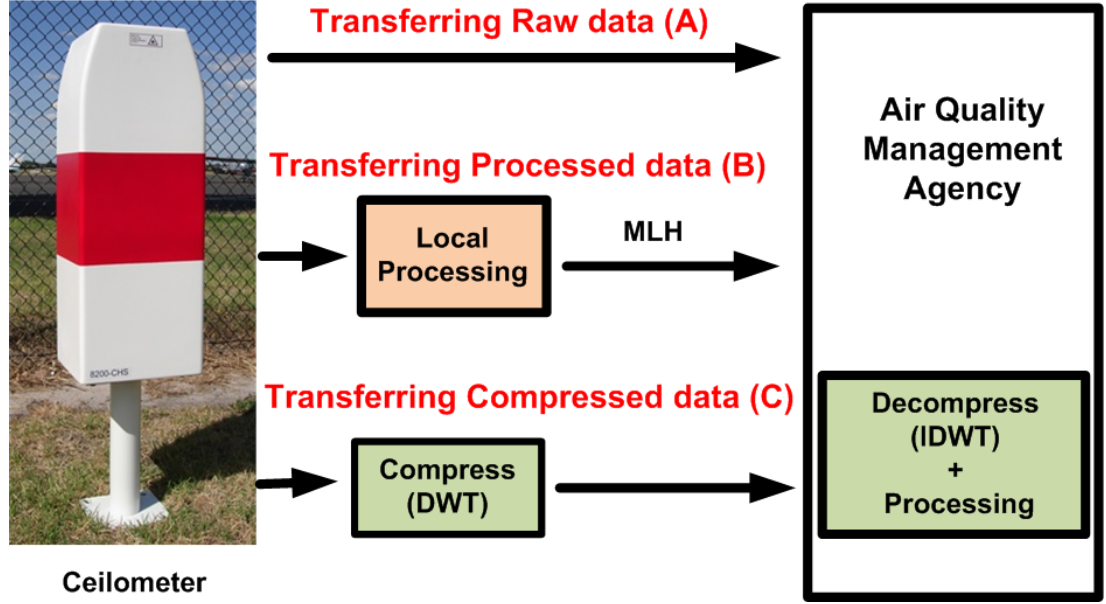


FIG. 5.3. Three different method for ceilometer data transfer (A) Transferring raw data over existing network, (B) Locally processing data and transfer key values, (C) Compressing data and transfer to air quality management agency (base) for reconstruction and processing. Here Compression is done with DWT and decompression is performed by IDWT operation

5.3 Framework and Algorithms

5.3.1 Framework

The conventional way to process LIDAR sensor data is to transfer the data to air quality management agency or base and perform data processing. However, since ASOS station transmission bandwidth is limited, raw data transmission over radio frequency bandwidth is not possible. We propose two solution to this problem (a) process data at sensor location and transfer key values over the existing network, (b) compress data at sensor location and transmit and then decompress the data at base and conduct processing, as shown in Figure 5.3 . We propose a framework for compression with Discrete Wavelet Transform

(DWT) and decompression with Inverse Discrete Wavelet transform (IDWT). Moreover, For MLH retrieval from profile, we proposed a computationally less expensive algorithm, suitable for embedded deployment and comparing results obtained from original and re-constructed profile. Furthermore, to achieve faster execution time and higher throughput we modify our DWT and IDWT algorithm with FFT based convolution and deploy it in embedded hardware.

5.3.2 Discrete Wavelet Transform (DWT)

Compression of the LIDAR profile is done with Discrete wavelet transform (DWT). In wavelet analysis, the DWT decomposes a signal into a set of mutually orthogonal wavelet basis functions. These functions differ from sinusoidal basis functions in that they are spatially localized that is, nonzero over only part of the total signal length. Furthermore, wavelet functions are dilated, translated and scaled versions of a common function known as the mother wavelet. Unlike the Discrete Fourier Transform (DFT), the DWT, in fact, refers not just to a single transform, but rather a set of transforms, each with a different set of wavelet basis functions. Two of the most common are the Haar wavelets and the Daubechies set of wavelets. DWT of a signal x is determined by feeding it to a series of filters. First the signal is fed simultaneously to a low pass (LP) filter and a high pass (HP) filter where g and h is the impulse response of the low and high pass filters respectively. Since each filter removes half of the frequencies by filtering, both filter response undergo downsampling by a factor of 2.

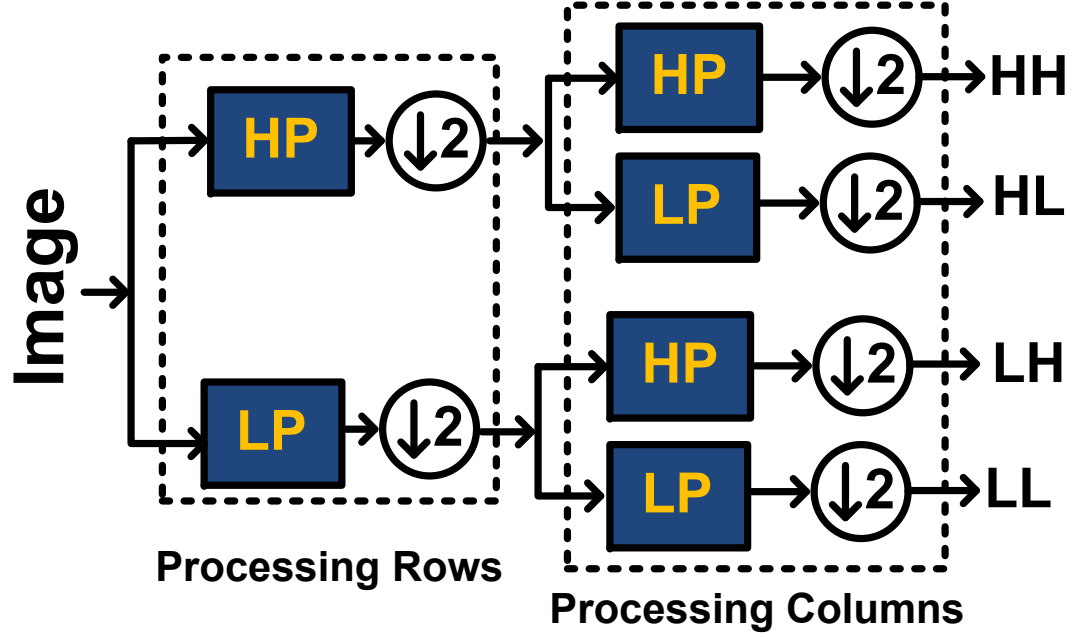


FIG. 5.4. compression using discrete wavelet transform. HP: High Pass, LP: Low Pass, HH: High pass-High pass, HL: High pass-Low Pass, LH: Low pass-High pass, LL: Low pass-Low pass

$$(5.1) \quad y_{lp}[n] = (x \star g) \downarrow 2 [n] = \sum_{k=-\infty}^{\infty} \left(x[k] g[2n - k] \right)$$

$$(5.2) \quad y_{hp}[n] = (x \star h) \downarrow 2 [n] = \sum_{k=-\infty}^{\infty} \left(x[k] h[2n - k] \right)$$

First, we apply a one-level, one dimensional DWT along the rows of the image. Second, we apply a one-level, one-dimensional DWT along the columns of the transformed



FIG. 5.5. (a) Original image (b) Four elements after decomposition consisting of approximate image, horizontal details, vertical details, and diagonal details

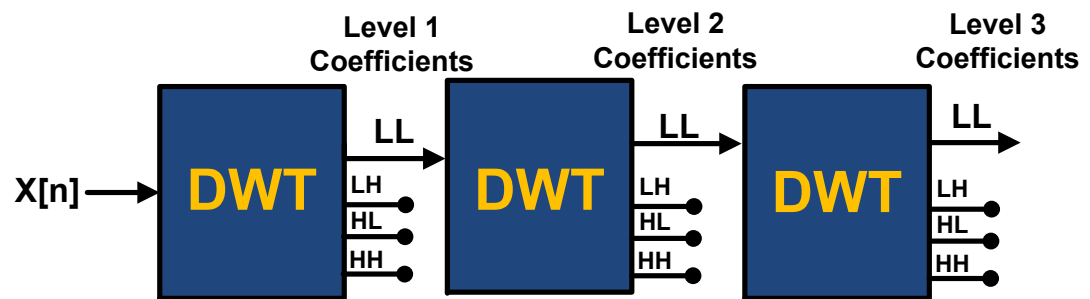


FIG. 5.6. Multilevel compression with DWT, where after each DWT operation, LL band is forwarded to the next DWT operation

image from the first step. The result of these two sets of operations is a transformed image with four distinct bands: (1) LL, (2) LH, (3) HL and (4) HH as shown in Figure 5.4. Here, L stands for low-pass filtering, and H stands for high-pass filtering. The LL band corresponds to an approximate image which is roughly down-sampled (by a factor of two) version of the original image. The LH band tends to preserve localized horizontal features, while the HL band tends to preserve localized vertical features in the original image. Finally, the HH band tends to isolate localized high-frequency point features in the image as shown in Figure 5.5 where the original lena image is decomposed into four components. If we transmit LL band from the sensor and discard the high frequency bands or details such as LH,HL, and HH and reconstruct the image only from the LL component, we can reduce transmission data-size by 75%. For multilevel compression, LL band from a previous DWT operation, is again fed to a DWT stage as shown in Figure 5.6 to achieve further compression.

5.3.3 Inverse Discrete Wavelet Transform (IDWT)

Decompression task is performed by Inverse Discrete Wavelet Transform (IDWT). In decompression stage, the four wavelet coefficients from DWT are used to reconstruct the signal. DWT coefficients are first up-sampled putting zero between every entries and thus doubling the size of the signal, which is then convolved with reconstruction scaling filter for approximating coefficient. The reconstruction scaling filter is built by flipping the filter coefficients. The results are summed for determining the original signal. For 2D signal,

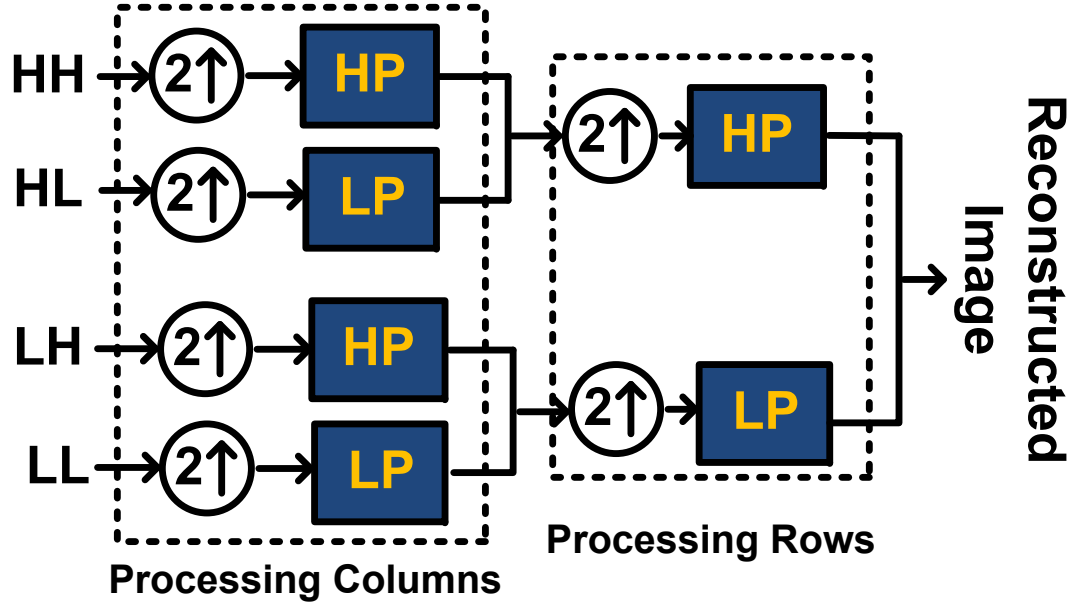


FIG. 5.7. decompression using inverse discrete wavelet transform

first columns are processed followed by processing on the rows as shown in Figure 5.7.

For our case, to achieve reduction in data size for transmission, we are only forwarding LL band to IDWT operation and initializing other three wavelet bands with zeros.

$$(5.3) \quad x_{lp}[n] = (y_{lp} \star g') \uparrow 2 [n] = \sum_{k=-\infty}^{\infty} \left(x[k] g'[n/2 - k] \right)$$

$$(5.4) \quad x_{hp}[n] = (y_{hp} \star h') \uparrow 2 [n] = \sum_{k=-\infty}^{\infty} \left(x[k] h'[n/2 - k] \right)$$

$$(5.5) \quad x_{reconst}[n] = x_{lp}[n] + x_{hp}[n]$$

5.3.4 FFT Based DWT and IDWT

DWT and IDWT can be considered as series filtering with different high pass and low pass filters where filtering involves convolution between the signal and a filter signal. To demonstrate efficiency of FFT based convolutions, the filtering mechanisms which is essentially Direct-Conv can be replaced by FFT based convolutions (FFT-Conv and FFT-OVA-Conv) as described in details in chapter 2.

5.3.5 Local Processing

We propose a less computationally intensive approach for MLH retrieval from LIDAR backscatter profile at sensor site. This approach consists of steps such as de-noising, feature enhancement and double derivative - as shown in Figure 5.8. After obtaining double derivative we perform a statistical average of the consecutive MLH. The parameters of de-noising, range and weight of feature enhancement and number of sequential elements to average are empirically determined for different dataset and sensor type.

$$(5.6) \quad G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

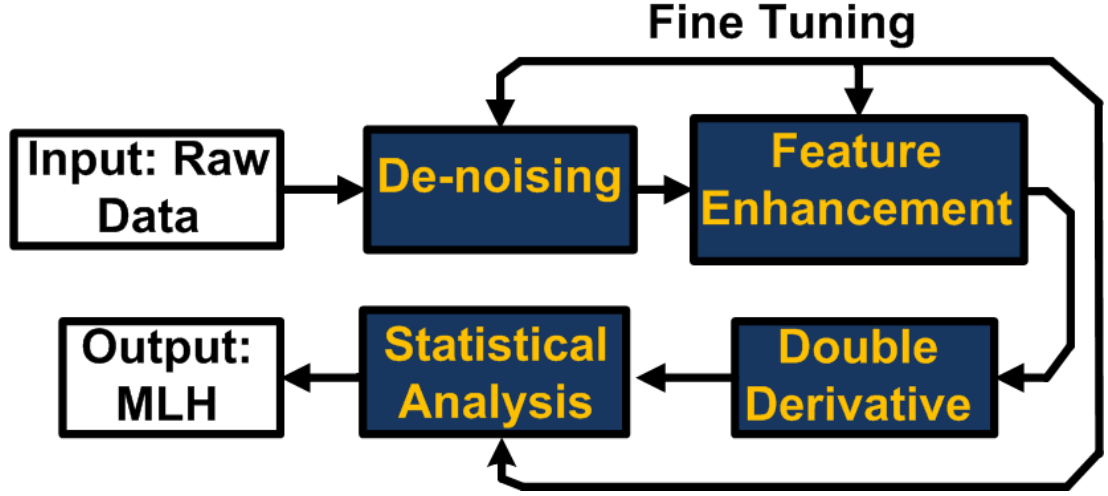


FIG. 5.8. Proposed prototypes for local processing algorithm

De-noising is performed by convolving the input with a Gaussian distribution $G(x,y)$. The standard deviation of the distribution σ is empirically obtained. Feature augmentation part consists of amplifying a range of input data from for better prediction of MLH. The parameters are chosen by trial and error.

5.4 Dataset

We use Vaisala CL31 ceilometer data from Plains Elevated Convection At Night (PECAN) campaign held in the summer of 2015. PECAN campaign was envisioned as a multi-agency project (NSF, NOAA, NASA, DOE) designed to advance the understanding of continental, nocturnal, warm-season precipitation. The LIDAR sensor CL31 was located in Fixed PISA 2 site at Greensburg, Kansas and gathered data from June 1 -July 15, 2015. Each day of data contains around 6000 time samples each containing 770 vertical

backscatter data.

5.5 Implementation Results

5.5.1 Simulation Results

Initial simulations for prototypes are done in matlab. Fig. 5.9 shows MLH retrievals from local processing algorithm run on LIDAR backscatter profile. The MLH results were cross checked with radiosonde launch. Original profile was compressed multiple times resulting into 75%, 93.8%, 98.4%, 99.6% and 99.9% percent of compression. Compressed profiles were reconstructed and local processing algorithm was run over for MLH retrieval. Fig. 5.10 shows MLH retrievals on different level of compressed profiles. We quantified the amount of detail loss from original profile due to compression and reconstruction through means squared error and Peak signal-to-noise ratio (PSNR). Fig. 5.11 shows MSE and PSNR over different level of compression. Level 1-3 achieves around 60 mse and increases afterwards. Level 1-4 retain 30 db PSNR and drops afterwards. Fig. 5.12 shows deviation between MLH retrieved from original profile and reconstructed profiles. The average deviation from Level 1 compression is 45m and level 2 is 55m respectively. Therefore both level 1 or level 2 compression can be selected for hardware deployment.

5.5.2 Hardware Results

Prototypes are deployed in quad core ARM cortex-A53 processor running at 1.2GHz frequency. Scripting was done with Octave GNU programming language with LTFAT de-

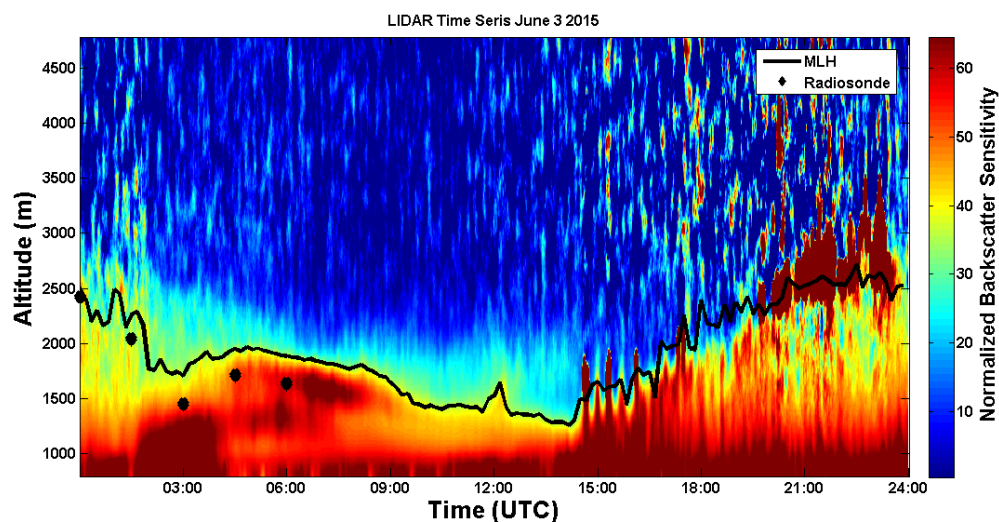


FIG. 5.9. Matlab simulation for MLH retrieval with proposed local processing algorithm with Radiosonde launch results

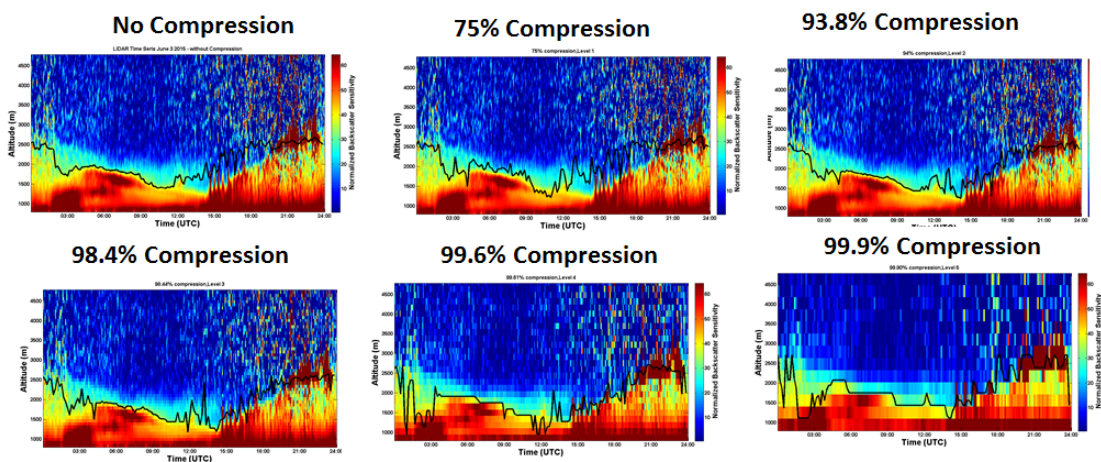


FIG. 5.10. MLH retrieval on original profile and multilevel reconstructed profile from 75%, 93.8%, 98.4%, 99.6% and 99.9% percent of compression

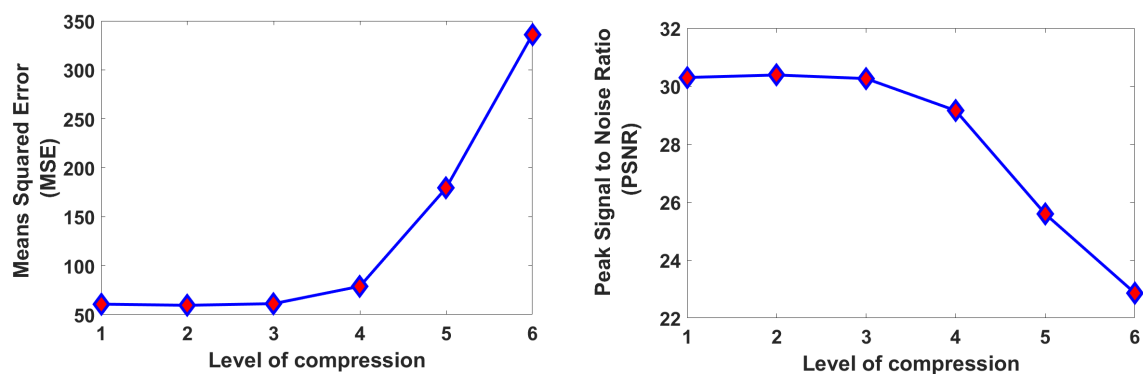


FIG. 5.11. Mean squared error (MSE) and Peak Signal to Noise Ratio (MSE) for reconstruction

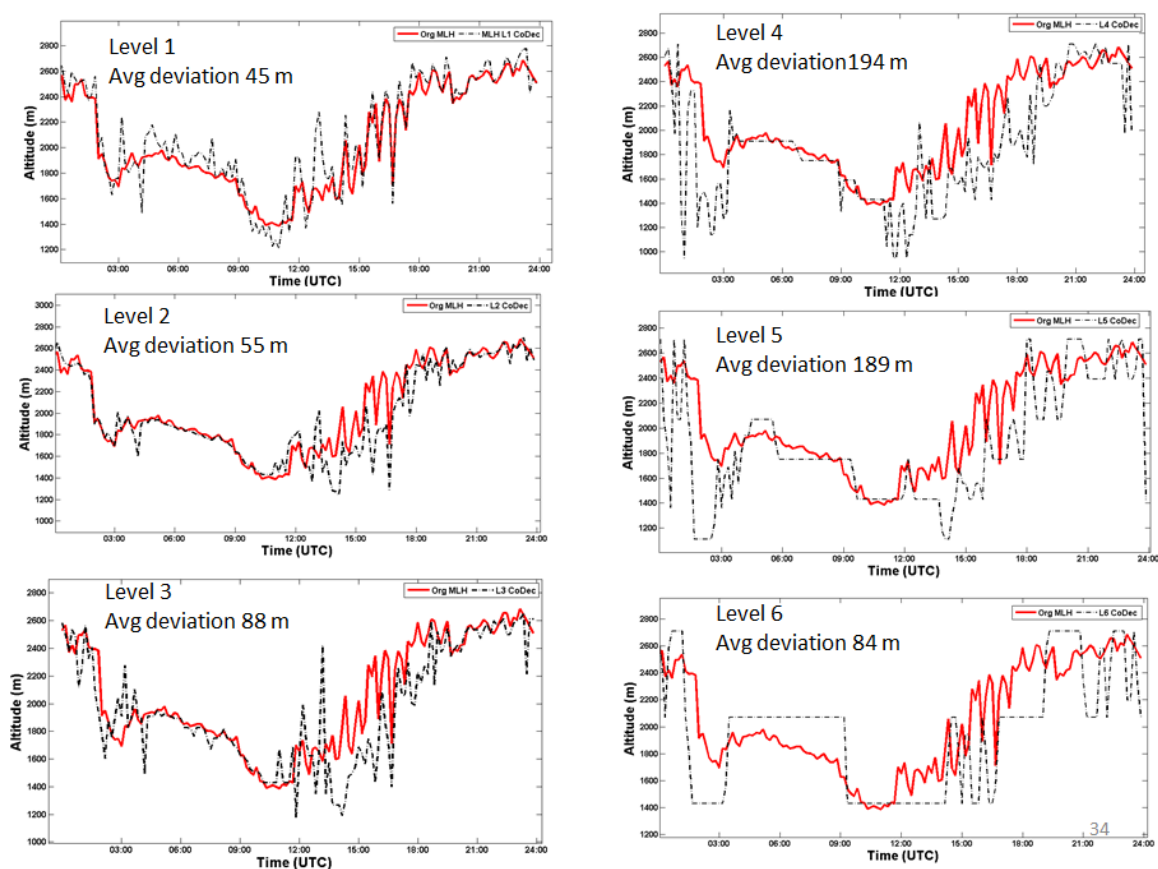


FIG. 5.12. Difference between MLH retrievals from original profile and multilevel reconstructed profile from 75%, 93.8%, 98.4%, 99.6% and 99.9% percent of compression

pendency. Timing analysis was conducted through octave's timing measurement functions. For prototyping we introduced 1 level of compression (75% compression) and subsequent decompression. Transmission scheme is bypassed for simplification and both compression and decompression is taken out in same ARM A53 CPU. Ideally we will deploy two ARM A53 CPU one compressing and transmitting signal through Wifi or blue-tooth system. Local processing a day of ceilometer data takes 4.43 min with throughput of 16.6 Ksample/s and 75% compression decompression takes 32.13s with throughput of 138.6 Ksample/s. FFT-OVA-Conv based CoDec achieved $10.6\times$ and $4\times$ faster execution time than Direct-Conv and FFT-Conv based CoDec method in ARM A53 CPU.

Results for local processing and different convolution based CoDec approach

| Process | Execution Time (s) | Throughput (Ksample/s) | Energy (J) |
|----------------------------|-------------------------------|-----------------------------------|-----------------------|
| Local Processing | 264 | 16.6 | 154 |
| CoDec (Direct-Conv) | 32 | 138.6 | 19 |
| CoDec (FFT-Conv) | 11.9 | 372 | 6.9 |
| CoDec(FFT-Ova-Conv) | 3 | 1478 | 1.7 |

Chapter 6

CONCLUSION

6.1 Results Summary

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, and ARM Cortex A53 CPU to explore the trade-off within software and hardware implementation, domain specific logic and instructions, as well as different parallelism across different architectures. For the case study of object detection, 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. PENC FFT-OVA-Conv has an execution time of 12.4 ms and average per layer throughput of 60.5 MB/s with 1.84 W average per layer power consumption and is $2.9\times$ and $1.65\times$ faster and achieves $6.7\times$ and $2.3\times$ higher throughput per watt than Direct-Conv and FFT-Conv respectively. On the ARM A53 CPU, the FFT-OVA-Conv achieves

3.36 \times and 1.38 \times improvement in execution time and achieves 2.72 \times and 1.32 \times higher throughput than Direct-Conv and FFT-Conv. On NVIDIA TX1 GPU FFT based convolution is 1.9 \times faster, 2.2 \times more energy efficient and achieves 5.6 \times higher throughput per layer than Direct-Conv. PENC is 10,916 \times and 1.8 \times faster and 9,200 \times and 7.9 \times more energy efficient and achieves 7.5 \times and 1.2 \times higher throughput per layer than ARM A53 CPU and TX1 GPU, respectively. For the case of atmospheric big data compression, FFT based DWT for compression decompression was proposed and implemented in ARM Cortex A53 CPU to solve ASOS's bandwidth limitation problem. Local processing of a day of ceilometer data took 4.43 min with a throughput of 16.6 Ksample/s and 75% compression decompression takes 32.1 s with a throughput of 138.6 Ksample/s. FFT-OVA-Conv based CoDec achieved 10.6 \times and 4 \times faster execution time than Direct-Conv and FFT-Conv based CoDec method in ARM Cortex A53 CPU.

6.2 Future Work

There are several options to extend this research. Firstly, the ResNet-20 trained on CIFAR-10 dataset is a less complicated version of object detection network. Implementing with deeper ResNets such as ResNet-56 or ResNet-110 trained with CIFAR-10 or ResNet-101 and ResNet-152 trained with ImageNet dataset will require more efficient memory and resource management modifications.

Secondly, we should explore different modifications of PENC many-core varying cluster memory size, router networks and communication protocols of loading and off loading

data to find the optimum embedded hardware suitable for CNN deployment.

Thirdly, we should design and build our FFT-based CNN system on FPGA and using cadence place and route tool design a chip.

Fourthly, for Atmospheric data transmission we should apply our FFT-based discrete wavelet transform for compression and decompression for other sensor data such as radiometers or wind profilers and explore whether it is beneficial.

Lastly, we have worked on developing a clustering based MLH retrieval method for LIDAR back-scattering profile. we should investigate more on the algorithm side and deploy on embedded hardware.

REFERENCES

- [Alwani *et al.*2016] Alwani, M.; Chen, H.; Ferdman, M.; and Milder, P. 2016. Fused-layer cnn accelerators. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 1–12.
- [Angevine, White, & Avery1994] Angevine, W. M.; White, A. B.; and Avery, S. K. 1994. Boundary-layer depth and entrainment zone characterization with a boundary-layer profiler. *Boundary-Layer Meteorology* 68(4):375–385.
- [Baars2007] Baars, H. 2007. Continuous monitoring of the planetary-boundary-layer depth with lidar. *Master’s thesis, Universität Leipzig, Germany*.
- [Balaji *et al.*2012] Balaji, P.; Buyya, R.; Majumdar, S.; and Pandey, S. 2012. 12th ieee/acm international symposium on cluster, cloud and grid computing.
- [Bertinetto *et al.*2016] Bertinetto, L.; Valmadre, J.; Henriques, J. F.; Vedaldi, A.; and Torr, P. H. S. 2016. Fully-convolutional siamese networks for object tracking. *CoRR* abs/1606.09549.
- [Bisasky, Chandler, & Mohsenin2012] Bisasky, J.; Chandler, D.; and Mohsenin, T. 2012. A many-core platform implemented for multi-channel seizure detection. In *Circuits and Systems (ISCAS), IEEE International Symposium on*, 564–567.

- [Bisasky, Homayoun, & others2013] Bisasky, J.; Homayoun, H.; et al. 2013. A 64-core platform for biomedical signal processing. In *Quality Electronic Design (ISQED), 2013 14th International Symposium on*, 368–372.
- [Collobert & Weston2008] Collobert, R., and Weston, J. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, 160–167. ACM.
- [Eaton et al.2014] Eaton, J.; Bateman, D.; Hauberg, S.; and Rik, W. 2014. *GNU Octave version 3.8.1 manual: a high-level interactive language for numerical computations*. CreateSpace Independent Publishing Platform. ISBN 1441413006.
- [Han et al.2016] Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M. A.; and Dally, W. J. 2016. EIE: efficient inference engine on compressed deep neural network. *CoRR* abs/1602.01528.
- [He et al.2015] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep residual learning for image recognition. *CoRR* abs/1512.03385.
- [Highlander & Rodriguez2016] Highlander, T., and Rodriguez, A. 2016. Very efficient training of convolutional neural networks using fast fourier transform and overlap-and-add. *CoRR* abs/1601.06815.
- [Ioffe & Szegedy2015] Ioffe, S., and Szegedy, C. 2015. Batch normalization: Ac-

celerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[Jafari & Mohsenin2015] Jafari, A., and Mohsenin, T. 2015. A low power seizure detection processor based on direct use of compressively-sensed data and employing a deterministic random matrix. In *IEEE Biomedical Circuits and Systems (Biocas) Conference*.

[Krizhevsky & Hinton2009] Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images.

[Krizhevsky, Sutskever, & Hinton2012a] Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012a. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

[Krizhevsky, Sutskever, & Hinton2012b] Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012b. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

[Kulkarni *et al.*2016] Kulkarni, A.; Abtahi, T.; Smith, E.; and Mohsenin, T. 2016. Low energy sketching engines on many-core platform for big data acceleration. In *Proceedings of the 26th Edition on Great Lakes Symposium on VLSI, GLSVLSI '16*, 57–62. New York, NY, USA: ACM.

[LeCun & Bengio1995] LeCun, Y., and Bengio, Y. 1995. Convolutional networks for

images, speech, and time series. *The handbook of brain theory and neural networks* 3361(10):1995.

[LeCun *et al.*1998] LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

[Lu *et al.*2015] Lu, J.; Young, S.; Arel, I.; and Holleman, J. 2015. A 1 tops/w analog deep machine-learning engine with floating-gate storage in 0.13 μm cmos. *Solid-State Circuits, IEEE Journal of* 50(1):270–281.

[Ma *et al.*2017] Ma, Y.; Cao, Y.; Vrudhula, S.; and sun Seo, J. 2017. End-to-end scalable fpga accelerator for deep residual networks.

[Mathieu, Henaff, & LeCun2013] Mathieu, M.; Henaff, M.; and LeCun, Y. 2013. Fast training of convolutional networks through ffts. *CoRR* abs/1312.5851.

[Ngiam *et al.*2011] Ngiam, J.; Khosla, A.; Kim, M.; Nam, J.; Lee, H.; and Ng, A. Y. 2011. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, 689–696.

[Ortega-Zamorano, Jerez, & Franco2014] Ortega-Zamorano, F.; Jerez, J. M.; and Franco, L. 2014. Fpga implementation of the c-mantec neural network constructive algorithm. *Industrial Informatics, IEEE Transactions on* 10(2):1154–1161.

- [Page *et al.*2016] Page, A.; Attaran, N.; Shea, C.; Homayoun, H.; and Mohsenin, T. 2016. Low-power manycore accelerator for personalized biomedical applications. In *Proceedings of the 26th Edition on Great Lakes Symposium on VLSI, GLSVLSI '16*, 63–68. New York, NY, USA: ACM.
- [Page *et al.*2017] Page, A.; Jafari, A.; Shea, C.; and Mohsenin, T. 2017. Sparcnet: A hardware accelerator for efficient deployment of sparse convolutional networks. *J. Emerg. Technol. Comput. Syst.* 13(3):31:1–31:32.
- [Park *et al.*2016] Park, S.; Park, J.; Bong, K.; Shin, D.; Lee, J.; Choi, S.; and Yoo, H. 2016. An energy-efficient and scalable deep learning/inference processor with tetra-parallel mimd architecture for big data applications. *IEEE transactions on biomedical circuits and systems*.
- [Qiu *et al.*2016] Qiu, J.; Wang, J.; Yao, S.; Guo, K.; Li, B.; Zhou, E.; Yu, J.; Tang, T.; Xu, N.; Song, S.; et al. 2016. Going deeper with embedded fpga platform for convolutional neural network. In *ACM International Symposium on FPGA*.
- [Rouhani, Mirhoseini, & Koushanfar2017] Rouhani, B. D.; Mirhoseini, A.; and Koushanfar, F. 2017. Tinydl: Just-in-time deep learning solution for constrained embedded systems. In *International symposium on circuits and systems (ISCAS)*. IEEE.
- [Rowley, Baluja, & Kanade1998] Rowley, H. A.; Baluja, S.; and Kanade, T. 1998. Neu-

- ral network-based face detection. *IEEE Transactions on pattern analysis and machine intelligence* 20(1):23–38.
- [Sermanet *et al.*2013] Sermanet, P.; Eigen, D.; Zhang, X.; Mathieu, M.; Fergus, R.; and LeCun, Y. 2013. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*.
- [Shen *et al.*2016] Shen, J.; Veddapunt, N.; Boddeti, V.; and Kitani, K. M. 2016. In teacher we trust: Learning compressed models for pedestrian detection.
- [Simonyan & Zisserman2014a] Simonyan, K., and Zisserman, A. 2014a. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- [Simonyan & Zisserman2014b] Simonyan, K., and Zisserman, A. 2014b. Very deep convolutional networks for large-scale image recognition. *CoRR* abs/1409.1556.
- [Sohn, Shang, & Lee2014] Sohn, K.; Shang, W.; and Lee, H. 2014. Improved multimodal deep learning with variation of information. In *Advances in Neural Information Processing Systems*, 2141–2149.
- [Srivastava *et al.*2014] Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.
- [Szegedy *et al.*2014] Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S. E.; Anguelov, D.;

- Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2014. Going deeper with convolutions. *CoRR* abs/1409.4842.
- [Szegedy *et al.*2015] Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2015. Rethinking the inception architecture for computer vision. *CoRR* abs/1512.00567.
- [Vasilache *et al.*2014] Vasilache, N.; Johnson, J.; Mathieu, M.; Chintala, S.; Piantino, S.; and LeCun, Y. 2014. Fast convolutional nets with fbfft: A gpu performance evaluation.
- [Vepakomma *et al.*2015] Vepakomma, P.; De, D.; Das, S. K.; and Bhansali, S. 2015. A-wristocracy: Deep learning on wrist-worn sensing for recognition of user complex activities. In *Wearable and Implantable Body Sensor Networks (BSN), 2015 IEEE 12th International Conference on*, 1–6. IEEE.
- [Wang *et al.*] Wang, M.; Xiao, T.; Li, J.; Zhang, J.; Hong, C.; and Zhang, Z. Minerva: A scalable and highly efficient training platform for deep learning.
- [Wu & Gu2015] Wu, H., and Gu, X. 2015. Towards dropout training for convolutional neural networks. *Neural Networks* 71:1–10.

