Sub-uJ Deep Neural Networks for Embedded Applications

Paul N. Whatmough[1,2], Sae Kyu Lee[2], Gu-Yeon Wei[2], David Brooks[2]

[1]Arm Research, Boston, MA
[2]Harvard University, Cambridge, MA
paul.whatmough@arm.com

***Abstract*—To intelligently process sensor data on internet of things (IoT) devices, we require powerful classifiers that can operate at sub-uJ energy levels. Previous work has focused on spiking neural network (SNN) algorithms, which are well suited to VLSI implementation due to the single-bit connections between neurons in the network. In contrast, deep neural networks (DNNs) are not as well suited to hardware implementation, because the compute and storage demands are high. In this paper, we demonstrate that there are a variety of optimizations that can be applied to DNNs to reduce the energy consumption such that they outperform SNNs in terms of energy and accuracy. Six optimizations are surveyed and applied to a SIMD accelerator architecture. The accelerator is implemented in a 28nm SoC test chip. Measurement results demonstrate ~10X aggregate improvement in energy efficiency, with a minimum energy of 0.36uJ/inference at 667MHz clock frequency. Compared to previously published spiking neural network accelerators, we demonstrate an improvement in energy efficiency of more than an order of magnitude, across a wide energy-accuracy trade-off range.**

Keywords—deep neural networks; accelerators; embedded; IoT; Razor; SoC

I. Introduction

The burgeoning internet of things (IoT) market segment has spurred interest in a varied assortment of new embedded computing applications and form factors. These new IoT applications often exploit an abundance of local sensor data. Machine Learning (ML) techniques enable us to interpret (noisy) sensor data, providing applications with a view into the real-world environment around them.

A wide range of classification tasks are becoming ubiquitous in embedded devices. Examples of these include: character classification in images, key-word classification in speech, and face detection in images. From an ML perspective, these kinds of tasks are often considered simple, if not trivial, and the research often focusses on much more challenging classification problems, such as large category image classification datasets, e.g. Imagenet. However, for heavily constrained embedded SoCs, even relatively simple neural network classifiers represent a serious challenge in terms of both compute load and memory footprint.

There has been a variety of work presented recently on specialized ML hardware, which has demonstrated the advantage of specialized designs compared to CPU or GPU implementations. These works tend to focus on either high-power convolutional neural networks (CNNs) for computer vision [1–3], or low-accuracy spiking neural networks (SNNs) [4–5]. For simpler classification tasks, it is still not clear which of a number of competing algorithms is the most efficient. SNNs have been strongly motivated, but deep-learning techniques can also be readily applied to constrained platforms too. SoCs for the IoT segment require balancing classification accuracy and energy to fall somewhere between the two extremes of CNN and SNN. To achieve this, it is necessary to systematically optimize across ML algorithms, computer architecture and digital circuits. In this paper, we will address each layer of this stack and present measured results on a 28nm SoC test chip, for a range of common classification problems, all at energy levels below 1uJ per classification [6].

The remainder of the paper is organized as follows. In section 2 we survey some background material related to deep neural networks and hardware accelerators. In section 3, we describe a series of optimizations that can be applied to neural networks to strongly reduce the energy consumption. Section 4 describes the proposed DNN ENGINE design, with implementation details. Section 5 presents measurement results. Finally, section 6 offers conclusions.

II. Background

A. Deep Neural Networks

Fully Connected Deep Neural Networks (FC-DNNs) are a mature ML model particularly well suited to general-purpose classification tasks [7–8]. Generally, FC-DNNs offer better accuracy than SNNs, but unfortunately do not map as well to VLSI implementations, as they present a much larger arithmetic workload and memory bandwidth requirement. An FC-DNN is typically represented as a simple weighted directed acyclic graph consisting of multiple layers of neurons (nodes) and weights (edges), as shown in Fig. 1. The *deep* designation refers to a graph containing more than one hidden layer, along with the input and output (softmax) layers. The additional hidden layers are an important characteristic, as they allow for much more complex non-linear functions to be learned. Concretely, the output of the j^{th} neuron in the k^{th} layer, $x_j(k)$ is given by

$$x_j(k) = \phi(\Sigma_i\, w_{ji}(k) \cdot x_i(k\text{-}1)), \quad (1)$$

where $w_{ji}(k)$ is a unique weight connecting neurons in the graph, and ϕ is the rectified linear (ReLU) activation function, $\phi(x) = \max(x,0)$. The $w_{ji}(k)$ *parameters* are learned during a *training* procedure [7], which is not described in this paper, as we are concerned solely with performing inference assuming a pre-existing trained model.

This kind of sum-of-products kernel (1) is very common in DSP algorithms. However, in FC-DNN, the kernel is typically much wider, and requires many more weights than might typically be employed in DSP algorithms, such as FIR filters. Nonetheless, as we will see, many of the optimizations that are common in implementing DSP algorithms are also relevant here.

B. Hardware Accelerators

At the cutting-edge, ML workloads are developing rapidly and flexible software approaches on SIMD CPUs and/or GPGPUs are the most practical approach. However, for embedded applications that have converged sufficiently, specialized hardware accelerators provide the efficiency needed in energy-constrained use-cases. Accelerators allow for much more freedom at implementation time, such that a number of optimizations can be exploited, that are otherwise difficult to implement in software. A more tailored implementation in custom hardware inevitably leads to much higher performance and efficiency. However, this comes at the cost of diminished programmability and can lead to premature obsolescence of the hardware platform.

Neural networks are a compelling target for hardware acceleration, because we can optimize the implementation for a single algorithm (e.g. FC-DNN in this case), but retain "programmability" for a range of applications, by changing the weights ($w_{ji}(k)$). Fig. 1 shows a typical application scenario for classifying sensor data, where provision for programmable weights and network topology allow for the freedom to update the classifier functionality as needs dictate. Although even greater efficiency could be achieved by hard-coding the weights for a given classification task, it would consume a very large amount of silicon area and would result in a design that can only be used for a single task.

III. Energy Optimizations

In this section, we describe a number of optimizations that can be applied to the computation of the FC-DNN compute graph to increase efficiency.

A. Parallelism

Deep neural networks are embarrassingly parallel. More specifically, within the k^{th} layer, there are no data dependencies whatsoever, since the weights, $w_{ji}(k)$ are unique and the neuron values, $x_j(k)$ are arranged in parallel in the graph. However, the degree of parallelism implemented is of course limited by the silicon area and by the memory bandwidth required to deliver the weights to the datapath. In practice, the memory bandwidth limitation is the most severe.

B. Data Reuse

An interesting side-effect of increasing the parallelism is that we see a benefit in terms of operand reuse. Operand reuse refers to the situation where we are able to load an operand from (local) memory and use it in more than one consecutive computation. In FC layers, there is no reuse in the weights at all, which are loaded, used once and then discarded.

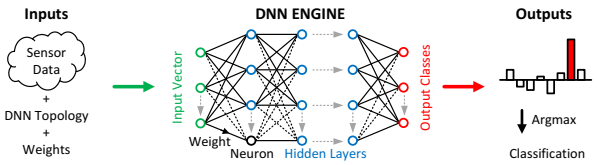

Fig. 1. Illustration of a hardware accelerator for the FC-DNN algorithm, which processes sensor data to generate a classification result. The topology and weights for the FC-DNN are programmable, which allows the design to be used for arbitrary classification tasks at run-time.

However, the activation data can be loaded and reused across all the parallel neurons in a given layer. This leads to significant savings in memory bandwidth, which ultimately result in energy savings. Although beyond the scope of this paper, CNNs exhibit reuse in both activations and weights, and hence 2D systolic array architectures are popular as they allow reuse of both of the MAC operands, rather than just one.

C. Sparse Data

Both weights and activations typically contain a large number of zero and small non-zero values. Since we are implementing a sum-of-products kernel, if either of the operands are close to zero, the resulting accumulator update will be negligible. Therefore, it is not necessary to perform operations associated with small operands, and we can skip them entirely. We exploit this property to reduce the compute requirement and memory bandwidth, without impacting the classification accuracy of the trained model.

In this work, we demonstrate dynamic activation pruning, which does not impose any new constraints on the DNN training procedure. The general approach is to threshold activation data as it is generated, and then only store significant activations that are larger than a pre-determined threshold. Subsequently, when calculating the next layer, we then only load the significant activations that were stored in the preceding layer. Operations and storage associated with small activations is pruned entirely, significantly reducing the memory bandwidth and also the compute.

D. Small Data Types

The optimal data type used in neural networks is the subject of significant research effort [7]. However, it is clear that for inference, it is possible to use quite small fixed-point types, without impacting classification accuracy. The bitwidths that can be safely used to represent the weights are typically in the range of 4-16 bits, and depend on the exact network that is being executed. In order to provide flexibility while also increasing efficiency, we support both 16-bit and 8-bit weights, with a number of programmable rounding modes, all of which are configurable on a per-layer basis for maximum flexibility.

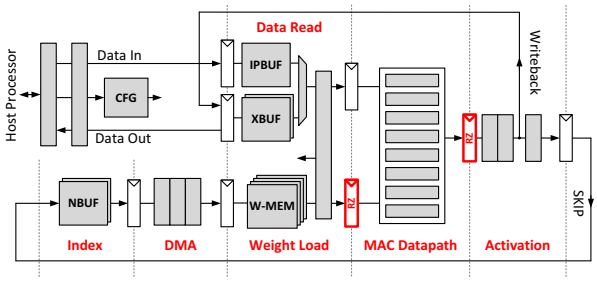

Fig. 2. Simplified micro-architecture of the DNN ENGINE, showing the pipeline stages and the interfacing.

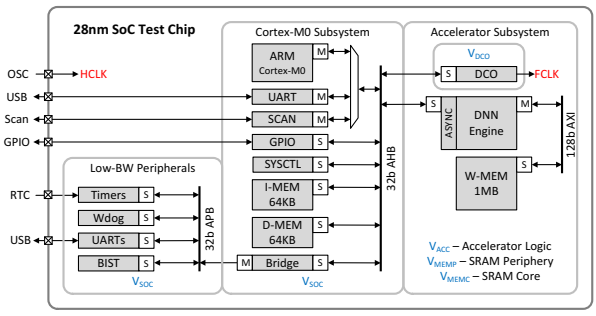

Fig. 3. Block diagram of the 28nm SoC test chip.

E. Noise Tolerance

Neural networks are inherently noise tolerant, and this can be exploited in a number of ways. The use of small datatypes, discussed previously, is an example of this. In this work, we also implement a light-weight Razor error detection scheme, which allows us to remove worst-case timing margins which account for process, voltage and temperature (PVT) variations. Razor is an approach that provisions for the occurrence of timing violations in order to detect and track changes in the PVT operating point [9]. Unlike previous work, even if timing errors are detected, it is not necessary to implement an expensive error correction mechanism, since DNNs can intrinsically tolerate noise arising from intermittent bit errors. A number of similar approaches have been previously studied for DSP algorithms, following either the Razor approach [10–12], or the earlier algorithmic noise tolerance (ANT) approach [13] pioneered by Shanbhag et al.

F. Weight Storage

Embedded devices typically have very limited access to bulk storage. When the system design does include off-chip flash or SRAM memory devices, it is typically very expensive to access them. This presents a challenge to the implementation of DNNs, since for inference, it is necessary to access the model weights, which tend to be of the order of hundreds of KBs for small classification problems. Accessing this through off-chip storage would lead to excessive energy consumption. Therefore, we implement an on-chip 1MB SRAM to store the weights, thereby eliminating off-chip access once the SRAM has been populated.

IV. DNN Engine

A. Design

The DNN ENGINE is a specialized hardware accelerator for efficiently performing FC-DNN inference for energy-constrained applications. The accelerator is programmable to allow processing FC-DNNs of various sizes for different application tasks and accuracy goals. The accelerator is controlled by a driver running on the host CPU, which configures the parameters of the target network, and stores the input data into a scratch pad memory. An address pointer is also provided for the network weights. The accelerator then processes the network one layer at a time, working on 8 neurons in parallel. After the output layer has been calculated, the accelerator sends an interrupt to the host CPU, which can retrieve the output data.

Fig. 2 is a simplified block diagram of the DNN ENGINE micro-architecture. The pipeline is arranged in five stages and incorporates all the optimizations described in section 3. The design is essentially based around a simple single-instruction multiple-data (SIMD) template. Hence, the datapath includes an 8-way MAC unit that calculates 8 neuron accumulations in parallel. The MAC datapath is fed with weight and activation operands from SRAM. The weights are stored in a 1MB SRAM on the SoC (outside the accelerator), while the activations are relatively small and stored in SRAM inside the accelerator itself. The activation stage adds a bias term and applies the ReLU activation function. It also applies a threshold function to test for sparse activations. Non-sparse activations are written back to XBUF for use in the following layer, while small activations are ignored. XBUF stores activation data, and is double-buffered to allow data for layer k to be read, while writing data for use in layer k+1. The DMA stage contains address generation logic used to sequence weights into the datapath in the correct order. Due to the sparse nature of the compute graph, the weight load sequences are not sequential and are derived from the contents of NBUF, which is a list of active nodes in the current layer and previous layers.

B. Implementation

The DNN ENGINE accelerator was implemented in a 28nm SoC test chip (Fig. 3). The SoC is based around an ARM M0 microcontroller, with associated peripherals and I/O blocks. The accelerator subsystem includes the DNN ENGINE macro, a 1MB SRAM block and a digitally controlled oscillator (DCO) to generate fast on-chip clocks. Various power domains are included to allow for a variety of voltage and clock frequency scaling experiments. Fig. 4 gives a summary and photo annotated with the floorplan for the SoC.

V. Results

Measured results for the DNN ENGINE demonstrate that on aggregate, the various optimizations result in an improvement in energy per inference at 667MHz, from 3.28uJ (0.9V, 16-bit), down to 0.36uJ (715mV, 8-bit, Razor, sparsity) – an improvement of nearly 10X. These results are for the MNIST dataset at 98.5% accuracy.

Process Tech.	TSMC 28nm HPC 1P10M
Die Size	2.4mm x 2.4mm
Total SRAM	SoC: 128KB / W-MEM: 1MB / DNN-Engine: 6.5KB
Total FFs (RZFFs)	8460 (896)
ML Model	FC-DNN Classifier
Weight Precision	8-bit / 16-bit Fixed-Point
Native Model Size	Hidden Layers: 0-6 Nodes/Layer: 1-1024
Error Tolerance	$>10^{-1}$ @ 98.5% Accuracy
Supply Voltage	0.6 – 1.1 V (operational)
F_{MAX}	667MHz @ 0.9V 1.2GHz @ 0.9V with Razor
Power Consumption	33.7mW @ 667MHz/0.9V/8b 20.3mW @ 667MHz/0.715V/8b 63.5mW @ 1.2GHz/0.9V/8b
Leakage	3.03mW @ 0.9V

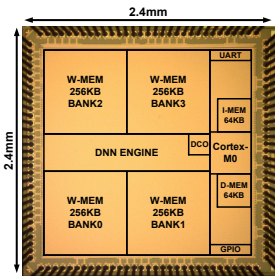

Fig. 4. Test chip summary table and die photograph.

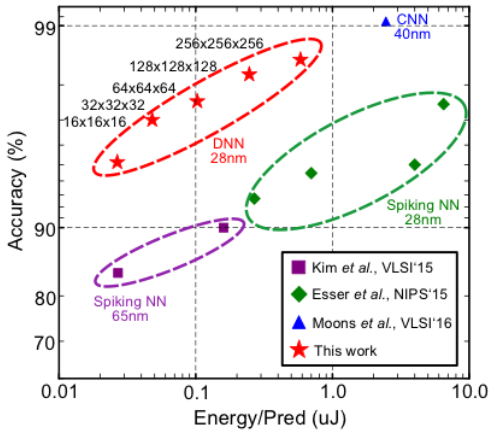

Fig. 5. Comparison of low-power hardware accelerators with published MNIST results. For the DNN ENGINE (this work), we present five different DNN topologies that cover a range of accuracy/energy points.

For energy constrained applications, there are a number of competing neural network models, namely DNN, SNN and CNN. Although SNNs have been strongly motivated as an energy efficient solution [4–5], it seems that they are best suited to moderate accuracy regimes, and even then, do not compete on energy with our optimized DNN implementation (Fig. 5). The DNN ENGINE demonstrates more than an order of magnitude improvement in energy efficiency. CNNs excel at image recognition tasks such as MNIST. Therefore, accuracy achieved is high. However, most CNN accelerators published to date are fairly powerful designs that are not well suited to energy-constrained devices. For example, they typically use off-chip memory which is not included in power measurements [1–3]. Hence, energy per prediction is higher.

VI. Conclusion

In this paper, we discussed the optimization of hardware accelerators for FC-DNN workloads. We described six areas for optimization, *viz.* parallelism, data reuse, sparse data, small data types, noise tolerance and weight storage. By exploiting these optimization opportunities, we implemented an accelerator architecture for energy constrained platforms. The accelerator is implemented in a 28nm SoC test chip, based around a microcontroller and associated peripherals. Measurement results demonstrate a reduction of nearly 10X in energy per inference due to the optimizations described. Compared to previously published spiking neural network hardware, the energy efficiency achieved is more than an order of magnitude higher.

Acknowledgment

This research was, in part, funded by the U.S. Government under the DARPA CRAFT (HR0011-16-C-0052) and PERFECT (HR0011-13-C-0022) programs. Intel Corporation also provided support. Arm Inc. kindly provided IP support.

References

[1] Y. H. Chen, et al., "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," IEEE Journal of Solid-State Circuits, Jan. 2017.

[2] B. Moons and M. Verhelst, "A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets," IEEE Symp. on VLSI Circuits, 2016.

[3] J. Sim, et al., "14.6 A 1.42TOPS/W deep convolutional neural network recognition processor for intelligent IoE systems," IEEE Int. Solid-State Circuits Conf. (ISSCC), 2016.

[4] J. Kim et al., "A 640M pixel/s 3.65mW sparse event-driven neuromorphic object recognition processor with on-chip learning," IEEE Symp. on VLSI Circuits, 2015.

[5] S. Esser, et al., "Backpropagation for energy-efficient neuromorphic computing", Int. Conf. on Neural Information Processing Systems, 2015.

[6] P. N. Whatmough, et al., "14.3 A 28nm SoC with a 1.2GHz 568nJ/prediction sparse deep-neural-network engine with >0.1 timing error rate tolerance for IoT applications," IEEE Int. Solid-State Circuits Conf. (ISSCC), 2017.

[7] B. Reagen, R. Adolf, P. Whatmough, G.-Y. Wei, D. Brooks, "Deep Learning for Computer Architects", Synthesis Lectures on Computer Architecture, Morgan & Claypool, August 2017.

[8] B. Reagen, et al., "Minerva: enabling low-power, highly-accurate deep neural network accelerators", Int. Symp. on Computer Architecture (ISCA), 2016.

[9] D. Ernst et al., "Razor: a low-power pipeline based on circuit-level timing speculation," IEEE/ACM Int. Symp. on Microarchitecture, 2003.

[10] P. N. Whatmough, S. Das and D. M. Bull, "A Low-Power 1-GHz Razor FIR Accelerator With Time-Borrow Tracking Pipeline and Approximate Error Correction in 65-nm CMOS," IEEE Journal of Solid-State Circuits, vol. 49, no. 1, pp. 84-94, Jan. 2014.

[11] P. N. Whatmough, S. Das, D. Bull and I. Darwazeh, "Error-resilient low-power DSP via path-delay shaping," ACM/EDAC/IEEE Design Automation Conference (DAC), 2011, pp. 1008-1013.

[12] P. N. Whatmough, S. Das, D. M. Bull and I. Darwazeh, "Circuit-Level Timing Error Tolerance for Low-Power DSP Filters and Transforms," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, no. 6, pp. 989-999, June 2013.

[13] R. Hegde and N. R. Shanbhag, "A voltage overscaled low-power digital filter IC," in IEEE Journal of Solid-State Circuits, vol. 39, no. 2, pp. 388-391, Feb. 2004.