An Energy-Efficient Precision-Scalable ConvNet Processor in 40-nm CMOS

Bert Moons, *Student Member, IEEE*, and Marian Verhelst, *Senior Member, IEEE*

***Abstract*— A precision-scalable processor for low-power ConvNets or convolutional neural networks is implemented in a 40-nm CMOS technology. To minimize energy consumption while maintaining throughput, this paper is the first to implement dynamic precision and energy scaling and exploit the sparsity of convolutions in a dedicated processor architecture. The processor's 256 parallel processing units achieve a peak 102 GOPS running at 204 MHz and 1.1 V. It is fully C-programmable through a custom generated compiler and consumes 25–287 mW at 204 MHz and a scaling efficiency between 0.3 and 2.7 effective TOPS/W. It achieves 47 frames/s on the convolutional layers of the AlexNet benchmark, consuming only 76 mW. This system hereby outperforms the state-of-the-art up to five times in energy efficiency.**

***Index Terms*— Approximate computing, ConvNet, convolutional neural network (CNN), deep learning, Dynamic-Voltage-Accuracy-Scaling, processor architecture, voltage scaling.**

I. Introduction

DEEP learning [1] networks and more specifically ConvNets or convolutional neural networks (CNNs) have come up as state-of-the-art classification algorithms, achieving near-human performance in applications in both computer vision (CV) and automatic speech recognition. ConvNets have been used to achieve unprecedented accuracy for tasks ranging from phone recognition [2], handwritten-digit recognition [3], [4], object recognition [5]–[8], object detection [9], [10], scene understanding or semantic segmentation [11], and even video recognition [12]. Although these networks are extremely powerful, they are also very computationally and memory intensive, requiring hundreds of megabytes for filter weight storage and hundreds of millions of operations per input. This high cost makes them difficult to employ on embedded or battery-constrained systems. However, the energy consumption of neural networks can be significantly reduced by exploiting a number of algorithm-specific observations.

First, hardware implementations of neural networks can be made more energy efficient by exploiting temporal and spatial data locality [13].

Manuscript received August 5, 2016; revised October 13, 2016; accepted November 26, 2016. Date of publication December 29, 2016; date of current version March 23, 2017. This paper was approved by Guest Editor Makoto Ikeda. This work was supported in part by Research Foundation – Flanders and in part by Intel Corporation.

The authors are with the Department of Electrical Engineering, ESAT-MICAS, KU Leuven, 3001 Leuven, Belgium (e-mail: bert.moons@esat.kuleuven.be; marian.verhelst@esat.kuleuven.be).

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

Digital Object Identifier 10.1109/JSSC.2016.2636225

Second, [14]–[17] show that ConvNets are inherently fault tolerant and can be operated at low computational precision with limited accuracy loss. Reference [15] even goes down to 1-b operations with more accuracy loss.

Third, as in [14], [18], and [19], ConvNets are extremely sparse and can be compressed to reduce their memory footprint and be made more efficient by skipping unnecessary computations.

Several works have proposed specialized and optimized CNN data flows for energy-efficient network inference, either on existing platforms or on novel hardware architectures. Optimizations for high-performance applications on CPU [20], GPU [21], or FPGA [22]–[24] all consume several to hundreds of watts, making them unusable in battery-constrained embedded systems. Other works [25]–[32] are application-specific integrated circuits (ASICs) that focus on low-power embedded applications, aiming to achieve real-time operation at subwatt power consumption. They are all accelerators, reducing their flexibility in exchange for energy efficiency. Eyeriss [25] proposes a 2-D spatial architecture, exploiting data locality and network sparsity, but does not exploit reduced precision computations. Reference [26] is an optimized architecture exploiting reduced precision, but can perform only a hardwired number of layers. DaDianNao [27] and ShiDianNao [28] exploit locality, but achieve high performance only for small neural networks. Reference [29] uses a specific 7×7 convolutional engine, which limits flexibility, operating at a constant 12-b fixed precision. Reference [30] combines DaDianNao's architecture with hardware support to exploit network sparsity in the time domain, achieving up to a $1.55\times$ performance improvement. EIE [31] and Minerva [32] exploit reduced precision and sparsity in a novel hardware architecture, but tailored only for fully connected (FC) network layers.

The work [33] outperforms the state-of-the-art works up to $5\times$ in energy-efficiency as it is the first to exploit all main energy-saving opportunities in ConvNets: 1) data locality; 2) precision scaling; and 3) network sparsity. These results were achieved through three key innovations.

1) A processor architecture employing a 2-D single-instruction multiple-data (SIMD) Multiply-Accumulate (MAC) array.
2) Hardware support for Dynamic-Voltage-Accuracy-Scaling, allowing modulation of both the computational precision and the used supply voltage from layer to layer. We hereby illustrate the concept of approximate computing [34]: an efficient baseline processor can be made more energy efficient if the algorithm requires less accurate computations.

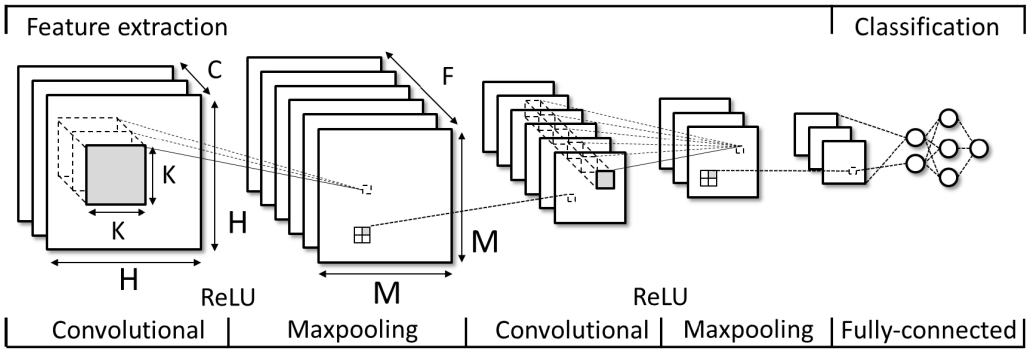


Fig. 1. Overview of a typical ConvNet architecture: two sequences of convolutional–ReLU–Maxpooling layers perform hierarchical feature extraction. This is followed by a multilayer FC classification stage.

TABLE I
PARAMETERS OF A CONV LAYER

Parameter	Description	Range
F	Number of filters per layer	16-512
H	Width & height of input feature map	16-227
C	Number of channels in input feature map	3-512
K	Width & height of filter plane	1-11
M	Width & height of output feature map	16-227

3) Hardware support for network compression and guarding sparse operations.

This paper is organized as follows. Section II discusses the basics of ConvNets and the possibilities for efficiency optimization. Section III discusses the processor architecture. Sections IV and V discuss the necessary extra hardware support and layout considerations to achieve efficiency gains. Section VI illustrates the processor's instruction set. Section VII shows extensive measurement results, showcasing the efficiency gains of the different applied techniques. Section VIII compares with the state of the art, and Section IX concludes this paper.

II. EMBEDDED CONVNETS

A. ConvNet Background

ConvNets, as in Fig. 1, are a type of artificial neural networks inspired by visual neuroscience. They are a cascade of multiple stacked convolutional, nonlinearity, and pooling layers used for feature extraction, followed by a smaller number of FC neural network layers used for classification. The weights of all stages in the cascade of convolutional network layers are trained to represent hierarchical features. In a face recognition example, the first layers in a network will train to recognize crude features, such as edges and contrast changes. Deeper layers will train to recognize higher order features such as noses, mouths, or eyes, while the deepest layers will eventually model faces. The number of cascaded stages in recent ConvNet models varies anywhere from 2 [3], typically 10–20 [6] to more than one hundred [8], ending with typically three FC layers [5].

A convolutional (CONV) layer, with the topology parameters listed in Table I and Fig. 1, transforms input feature maps (I) into output feature maps (O), each containing multiple units. Each unit in an output feature map ($M \times M \times 1$) is connected to local patches of units ($K \times K \times C$) in the input feature maps through a filter bank (W) ($K \times K \times C \times F$) existing out of a set of machine-learned weights and a bias (B) per output feature map. All units from a single-output feature map share the same filter bank, while different feature maps in a layer use different filter banks. The basic computation of such a CONV layer is shown in Fig. 1. Using the shape parameters listed in Table I, a formal mathematical description is given in the following equation:

$$O[f][x][y] = \sum_{c=0}^{C} \sum_{i=0}^{K} \sum_{j=0}^{K} I[c][Sx+i][Sy+j] \times W[f][c][i][j] + B[z] \quad (1)$$

where S is a stride and x, y, and f are bounded by: $x, y \in [0, \ldots, M[$ and $f \in [0, \ldots, F[$.

The result of the local sum computed in this filter bank is then passed through a nonlinearity layer. In ConvNets, this layer is typically a rectified linear unit (ReLU), using the nonlinear activation function $f(u) = max(0, u)$, where u is a feature map unit. This activation function reduces the vanishing gradient problem [35] in the backpropagation-based training phase of the network and leads to high degrees of sparsity due to nonactivated outputs.

Maxpooling layers compute and output only the maximum of a local (typically 2×2 or 3×3) patch of output units in a feature map. They thereby reduce the dimension of the feature representation and create an invariance to small shifts and distortions in the inputs.

Finally, *FC layers* are used as classifiers in the ConvNet algorithm. An FC layer is mathematically described as the matrix vector product $O[z] = \sum_{m=0}^{M} W[z, m] \times I[m]$, where M is the size of vectorized input feature map and $z \in [0, \ldots, Z[$ is the number of neurons in the FC layer. As all the used weights are only used once in a forward pass, there is no weight reuse in these layers. Due to this observation, architectures proposed for FC layers, as in [31] and [32], are different from architectures for CONV layers.

As illustrated in Table II, the total network weight size is dominated by the FC layer weights (>90% of the total

TABLE II
MODEL SIZE AND COMPUTATIONAL COMPLEXITY COMPARISON BETWEEN THE FC AND CONV LAYERS

Network	CONV size $[\#w]$	FC size $[\#w]$	CONV ops $[\#MAC]$	FC ops $[\#MAC]$
LeNet-5 [3]	25.5k	405k	1888k	405k
AlexNet [5]	2.3M	58.6M	666M	58.6M
VGG-16 [6]	14.7M	124M	15.4G	124M
SqueezeNet [18]	733k	0	746M	0

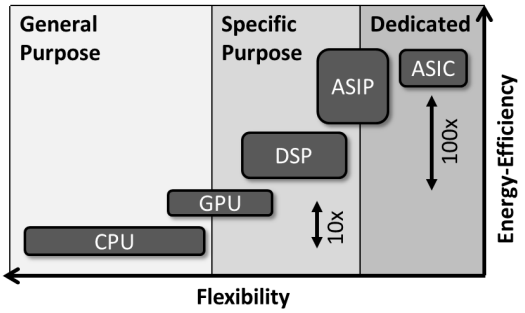


Fig. 2. Different hardware platforms in the energy efficiency versus flexibility space. ASIPs offer better flexibility than hardwired or reconfigurable ASICs at a similar energy efficiency.

network size), while the amount of computations is heavily dominated by the CONV layers (>90% of the total amount of MAC operations).

Even though FC layers dominate the network size, the amount of weights can be pruned down to 1%–5% of the original size [36], by explicitly removing unnecessary connections, leading to a significant speedup. As there is no such impressive compression or speedup possible for CONV layers, acceleration of these layers is the primary focus of the design presented in this paper.

B. Increasing Energy Efficiency in ConvNets

There are three distinct ways to minimize energy consumption in ConvNet acceleration.

1) Algorithm Optimized Hardware Architectures: Fig. 2 gives an overview of different hardware platforms for ConvNet implementations in the efficiency-versus-flexibility space. CPU platforms, are highly flexible, but very inefficient, as they are sequential and waste too much of their power budget in control overhead. CONV operations, which are inherently parallel, can be operated on parallel platforms, such as GPUs or naive SIMD DSP processors. These trade flexibility for energy efficiency, but they do not sufficiently exploit data locality and require too high bandwidth from DRAM and on-chip SRAM [13]. A hardwired solution, typical of ASICs, can be parallelized while exploiting data locality. However, inflexible solutions are not suited for ConvNet acceleration because of the vast amount of different ConvNet topologies. Table I shows the possible range of shape configurations in typical ConvNets.

This paper hence proposes the design of a C-programmable application-specific instruction set processor (ASIP), completely optimized for ConvNet data flows while maintaining flexibility.

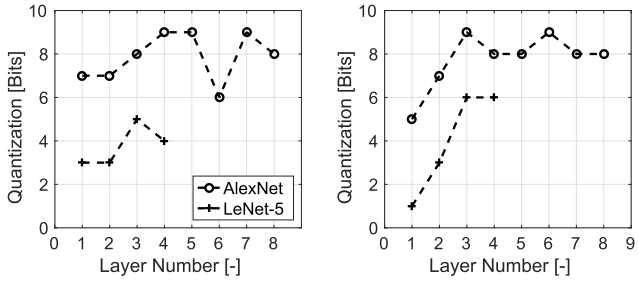


Fig. 3. Necessary number of bits for (a) weights and (b) input feature maps. With these quantization settings [14], the network achieves 99% relative accuracy compared with the full-precision 32-b floating point operation.

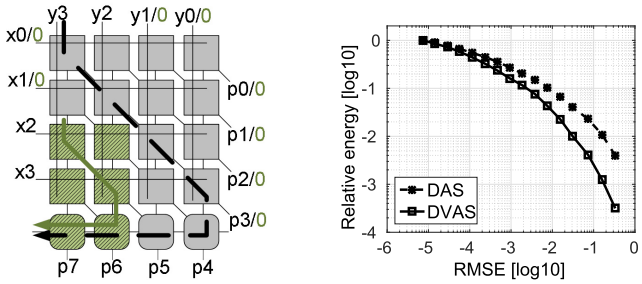


Fig. 4. (a) Basic principle of DVAS in a digital Baugh–Wooley multiplier example. (b) When precision is reduced by signal-gating LSBs, both the switching activity (Dynamic-Accuracy-Scaling (DAS)) and the critical path length are reduced, allowing for a reduced supply voltage for the whole multiplier. The combination of reduced α and V leads to major energy savings at limited root-mean-square error.

2) Dynamic-Voltage-Accuracy-Scaling: Typically, ConvNets are computed using a 32-b floating point precision [15]–[17]. However, they can be operated at fixed-point precision, at lower energy per operation. Reference [14] even shows that precision should not be fixed for a whole network, but can be modulated on a per-layer basis. Fig. 3 shows that 1- to 6-b precision suffices for LeNet-5 [3] on MNIST [37] data and 5- to 9-b precision for AlexNet [5] on ImageNet [38] data at 99% relative benchmark accuracy compared with a 32-b floating point baseline.

A multiplier capable of modulating its precision and supply voltage was first proposed in [39]. The simplified example in Fig. 4 can operate from 1 to 4 b, with two operating modes (2and 4b) highlighted in the figure. At high precision, all building blocks will be switching, resulting in a high switching activity and a long critical path (see the dashed line in Fig. 4). In the low-precision case, only the two most significant bits (MSBs) are used, resulting in a lower switching activity and shorter critical path (see the full line in Fig. 4). The LSBs are signal gated. At constant frequency, this shorter critical path can be compensated for through a lower supply voltage. Modulating the supply voltage from layer to layer is feasible, as a typical ConvNet layer takes $O(ms)$ or $O(us)$ to complete, while existing dc–dc converters for fast voltage scaling show transition times ranging from tens of nanoseconds [43] to subnanoseconds [44].

The combined effects:1) reduction of switching activity and 2) shorter critical paths allowing lower supply voltages have

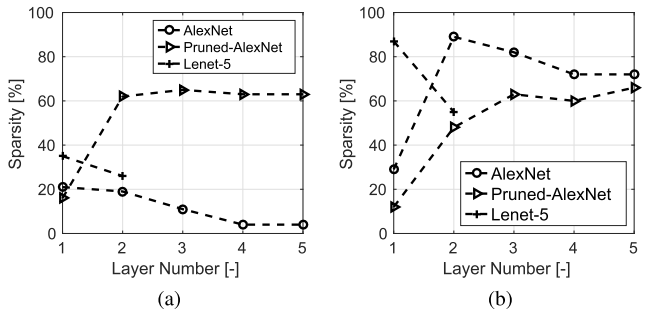


Fig. 5. Sparsity in the CONV layers for different benchmarks in the (a) weights and (b) input feature maps. AlexNet is optimally quantized, as in Section II-B2. Pruned-AlexNet is at full precision, as observed from [19].

a major impact on the system's dynamic power consumption. If leakage power is neglected, the power P of a DVAS system is given by

$$P = \alpha C f V^2 \rightarrow P = \frac{\alpha}{k_1} C f \left(\frac{V}{k_2}\right)^2 \quad (2)$$

where α is the circuit's switching activity, C is the technology dependent circuit capacitance, f is the clock frequency, and V is the system's supply voltage. At low precision, k_1 is a circuit architecture-dependent parameter and k_2 depends both on circuit architecture and technology.

This technique was first introduced as DVAS (DAS without voltage scaling) [39], in which the supply voltage can be modulated with varying precision requirements. The concept is similar to Dynamic-Voltage-Frequency-Scaling [40], in which the supply voltage is modulated with varying throughput requirements.

One disadvantage of DVAS is that it cannot be applied to all building blocks in a typical digital system. The critical paths of many building blocks, such as SRAM memories or control, decode, and fetch units, do not scale with reduced precision. Therefore, a DVAS system will be split into several power domains: one for all arithmetic capable of DVAS and one for all other building blocks. The design specifics for such a setup are discussed in Section IV.

3) Exploiting Network Sparsity: A final opportunity for energy reduction in embedded ConvNets is their high inherent sparsity. If weights or inputs are zero, their computations become redundant. Energy consumption can hence be reduced by explicitly skipping these computations.

Typical sparsity levels from [14] and [19] are shown in Fig. 5. Sparsity levels of up to 90% in the feature maps and up 20% in the network weights are measured for AlexNet performed on ImageNet. Sparsity can be due to three reasons. First, in reduced precision, small values will be explicitly mapped to integer zero. Second, ReLU layers map all negative input explicitly to integer zero. Finally, sparsity can also be explicitly enforced by pruning unimportant network connections [19]. Section V discusses the added hardware support to allow exploiting sparsity.

III. 2-D MAC Processor Architecture for Embedded ConvNets

The proposed programmable and energy-efficient ASIP architecture shown in Fig. 6 employs a 2-D SIMD MAC array as an efficient convolutional engine (Section III-A). For flexibility, configurable on-chip memory architecture and on-chip direct memory access (DMA) controllers (Section III-B) have been added. Section VI discusses the processor's instruction set.

TABLE III
Words Fetched per MAC Unit per MAC Operation

Filter Size	1D-SIMD	2D-SIMD	2D-FIFO	Gain [×]
1×1	2	0.125	0.125	16.0
3×3	2	0.125	0.086	23.3
5×5	2	0.125	0.078	25.6
11×11	2	0.125	0.072	27.8

A. Processor Datapath

1) 2-D MAC Array: The 16×16 2-D MAC array shown in Figs. 6 and 7 operates as a convolution engine. First, the array of single-cycle MAC units achieves a 256× speedup compared with a scalar solution, while minimizing the bandwidth to on-chip memory. The 2-D architecture allows applying 16 different filters to 16 different units of the input feature map simultaneously. This exploitation of data reuse thus allows a 256× speedup, at more than 16× lower internal bandwidth compared with a naive 1-D SIMD solution, requiring two inputs per processing unit per cycle. The local communication overhead can be further reduced by adding a first-in first-out (FIFO) register at the input of the MAC array, as shown in Fig. 7. Table III shows a comparison of the bandwidth fetch reductions of the 2-D MAC array architecture compared with the naive 1-D SIMD baseline.

The example in Fig. 7 illustrates the four first operation steps of a typical $3 \times 3 \times C$ filter data flow, of which four out of F filters are performed in parallel. The concept is illustrated for a simplified 4×4 MAC array for clarity, but the chip has a 16×16 array. In the first step, a vector is loaded from the input feature map buffer. This vector is stored in a FIFO register, multiplied with the first filter values of four different filters, accumulated with the previous result and stored in the accumulation register. In the second and third steps, a single word is fetched from the input feature map memory and pushed through the FIFO. This shifted vector is again multiplied with the next four filter values and accumulated with the previous result. This sequence is repeated three times for the 3×3 filter, illustrated by step 4 of Fig. 7, in which a vector of the next input row is multiplied and accumulated with the correct weights.

An additional advantage of this scheme is that it allows all intermediate values to be kept in the accumulation registers for a full $K \times K \times C$ convolution. Therefore, the MAC accumulation registers are 48 b, which is an overdesign for the worst case in the AlexNet benchmark. However, there is no need for frequent write-backs to SRAM. Here we leverage both spatial and temporal data locality [13], as we keep data local to minimize data movement and multiply and accumulate multiple weights with multiple inputs simultaneously.

2) 1-D SIMD Unit: The processor also contains a 1-D SIMD processing unit capable of performing multiple types of vector operations. This vector unit contains 16 parallel

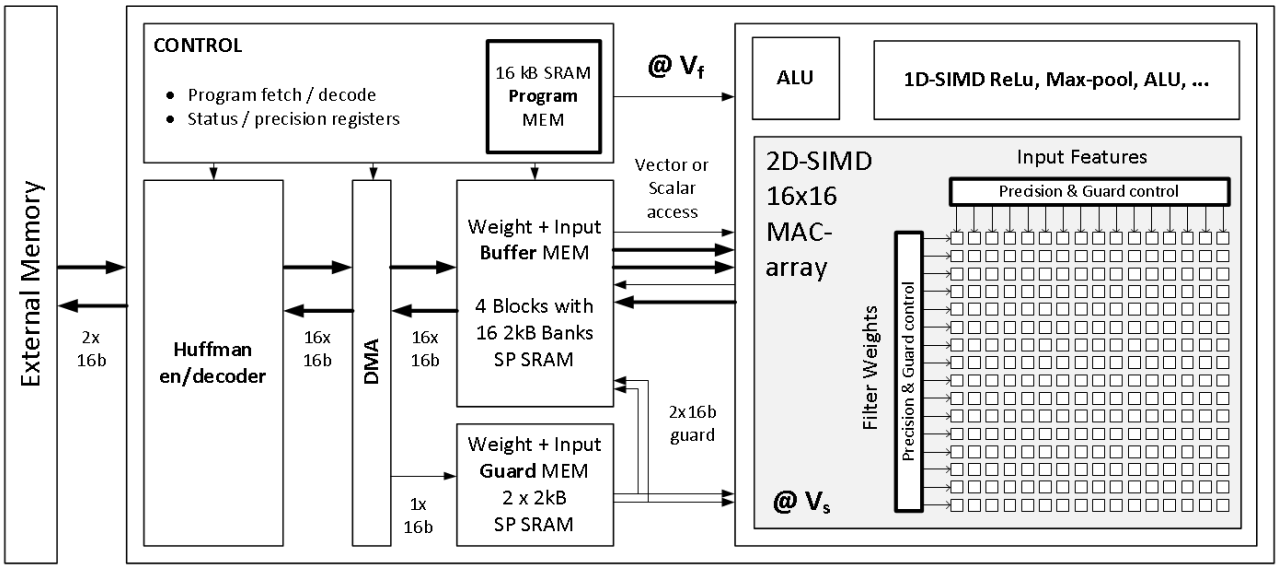


Fig. 6. High-level overview of the processor. The chip contains a scalar and 16× vector ALU, an input and a weight on-chip SP SRAM, an SRAM storing sparsity information, a control unit, and a DMA and a Huffman-based encoder/decoder in a fixed power domain at supply V_f. A 2-D SIMD MAC array is put in a scalable power domain at supply V_s.

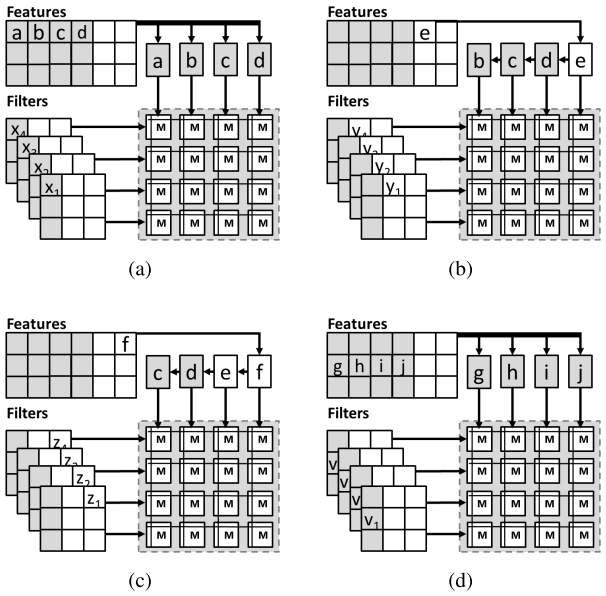


Fig. 7. Example of a typical ConvNet dataflow in the processor. In step 1, a vector of three input feature map units a, b, and c is fetched and multiplied with a vector of filter weights. The results are accumulated with the intermediate result that is stored in the MAC's accumulation register. In steps 2 and 3, a single feature d is fetched and pushed through the FIFO structure at the feature side. The resulting FIFO vector is multiplied and accumulated with a new weight vector. This sequence is repeated three times in this example, once for every row of the filter. (a) Step 1. (b) Step 2. (c) Step 3. (d) Step 4.

processing units with support for bitwise and shift operators, MAC, and Maxpooling (2×2 and 3×3).

3) Scalar Unit: The processor also contains a standard scalar ALU and MAC.

B. On-Chip Memory Architecture

1) On-Chip Main Memory: Fig. 8 shows the on-chip data memory topology. It is organized as one large 16-b memory

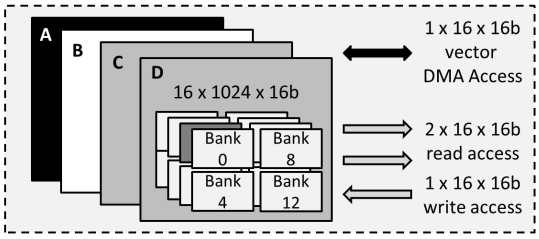
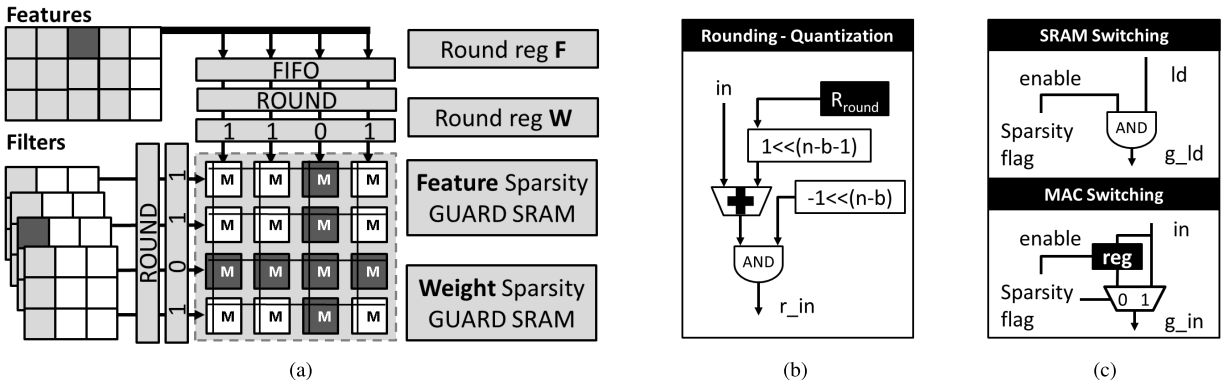


Fig. 8. On-chip memory architecture. One large address space is subdivided into four blocks, each containing 16 banks of single port SRAM. Every SRAM macro can store 1024×16 b words. The architecture allows scalar, vector, or double vector access from the processor side, while being read or written through a vector port from the DMA side.

address space (128 kB) and is subdivided into four blocks (32 kB) containing 16 single port SRAM banks (2 kB) of 1024×16 b words. Every block hence allows vector access, reading/writing one word from every bank. Single word access is also possible for scalar operations and for the FIFO-based architecture. The programmer is free to store feature maps or filter bank weights in any of the four available memory blocks. This cache architecture exploits temporal data locality [13]: it allows storing data close to the computational units for later reuse.

From the processor side, two of the four blocks can be read simultaneously. Typically, one block would contain only filter values and a second block a part of an input feature map. This allows fetching both filter and feature inputs for the 2-D MAC array in a single cycle. There is only one write port from the processor side due to the lower amount of write accesses.

2) Direct Memory Access Controller: A custom designed DMA allows efficiently communicating with off-chip memory, without stalling the processor pipeline. In parallel with access from the processor, any of the memory blocks can be read or written by the DMA, which is controlled by memory-mapped registers. Synchronization between the processor and the



(a) (b) (c)

Fig. 9. 2-D array is expanded with programmable rounding and memory fetch- and operator-guarding support. (a) Inputs of the array are rounded and guarded through arrays of (b) programmable rounding units and (c) guarding units. In (a), guard flags are indicated: they prevent two SRAM banks and a row and a column of MAC's from switching. In (b), n and b are the maximum (16) and the actually used number of bits (1–16). In (c), sparsity flag is used as an enable signal. $g_$ld and $g_$in are, respectively, the guarded load and MAC array inputs.

DMA is done by checking DMA-specific status registers, indicating if data transfer is done or not.

Furthermore, it contains a specific Huffman-based encoder/decoder [41] performing IO compression on sparse input and output data (Section V-B). Communication with the outside world is done using a 32-b parallel interface, transferring two words per cycle in standard mode.

IV. Hardware Support for Dynamic-Voltage-Accuracy-Scaling

The processor is made DVAS compatible by adding RTL support for precision scaling and by splitting the design into a scalable power domain for the MAC array and a fixed power domain for all other blocks. An advanced backend place and route optimization is needed as well.

A. RTL Level Hardware Support

The inputs to the MAC array in the scalable voltage domain are precision scaled by rounding their values to a programmable number of MSBs. The LSBs are explicitly gated to zero, to reduce switching activity. This is done through classical round-up rounding, transforming the positive number 10010 to 10100 if it is quantized to a 3-b MSB. This process requires two additions to the processor architecture.

First, two programmable status registers are added, as in Fig. 9(a), one for feature map inputs and one for filter weights, containing the number of bits that will be used for computation. These status registers can be written from software with a latency of 2 cycles. In practice, the precision is only changed on a per-layer basis. Second, two 1×16 vector arrays of programmable rounding units have to be added at the input of the MAC array, as shown in Fig. 9(a). Fig. 9(b) shows the building blocks of this rounding unit: one adder, two shifters, and 16 AND gates. Each programmable rounding unit is 267 NAND-2 equivalent gates.

The 2-D architecture is crucial to limit the overhead of the rounding units. Only $2 \times 16 = 32$ inputs have to be rounded in order to provide rounded inputs to $16 \times 16 = 256$ MAC units.

TABLE IV
Multimode Optimization Settings

Mode	Fixed V	Fixed f	Scalable V	Scalable f
4b	1.1V	200MHz	0.9V	400 MHz
8b	1.1V	200MHz	0.9V	200 MHz
12b	1.1V	200MHz	1 V	333 MHz
16b	1.1V	200MHz	1.1V	200 MHz

B. Physical Implementation

Creating a multipower-domain DVAS design, with one fixed and one scalable power domain for the MAC array, implies several complications in the physical implementation. Not only should the layout of the floorplan be adapted to contain level shifters, multiple power grids, and multiple supply ports, the backend optimization flow should be changed as well.

DVAS is not automatically possible if the backend optimizations are only performed in full precision mode. In a real chip, there are no guarantees that critical paths actually decrease at low computational precision, even if the circuit architecture does permit it. Theoretically, shorter paths in a multiplier can still be critical at low precision due to suboptimal placement, high wire delays, or small transistor sizings. Therefore, in DVAS, it is necessary to perform an advanced multimode multicorner place and route optimization scheme enforcing the theoretical potential of shorter critical paths at lower precision in the physical implementation. Ideally, the layout should be optimized for the continuous precision range from 1 to 16 bit, using libraries characterized at all supply voltages. However, satisfactory results were achieved optimizing only for the 4-, 8-, 12-, and 16-b modes. The design was optimized for 200-MHz operation in different simultaneously performed optimization modes, as listed in Table IV. If libraries were not available at the necessary voltage, the circuit was optimized at the lowest available voltage with heavier timing constraints, effectively mimicking lower voltage operation.

V. Hardware Support for Exploiting Network Sparsity

Hardware extensions are implemented to perform operation guarding and data compression.

A. Guarding Operations

At the start of a new CONV layer, both the sparsity information of the input feature map and of the layer filter banks is known. This information is stored in two dedicated small on-chip SRAM buffers [Fig. 9(a)], containing 1024×16 b words each (4-kB total), storing the sparsity information for one main memory block. Every word is a 16×1 b flag, where every flag denotes the sparsity information of an associated feature map unit or filter weight. If the flag is 1, the associated word contains valid data, if it is 0, it contains zero data. These flags are used for guarding both unnecessary memory fetches from and switching in the MAC array.

1) Guarding Memory Fetches: Guarding memory fetches is done by checking the sparsity flags before performing a fetch from on-chip memory. Depending on the value of the flag, words are conditionally fetched. This is simply done by gating the enable signals to the on-chip SRAM memories, as in Fig. 9(c). $16+16$ 1-b flags are checked in order to potentially prevent 32 large SRAM banks from switching.

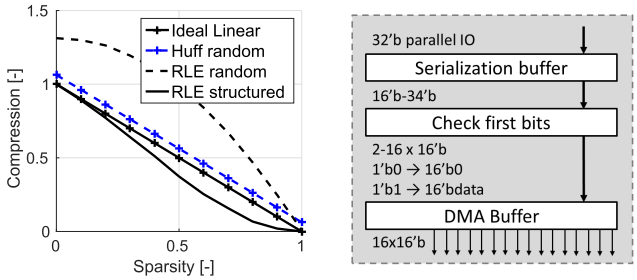
2) Guarding 2-D MAC Operations: Guarding 2-D MAC operations is done by both preventing the MAC inputs from switching and by clock gating the accumulation registers. To this end, extra arrays of switch prevention circuitry, as shown in Fig. 9, are added. A sparsity-guard register is controlled by a sparsity flag. If the flag is 1, a new input value is stored in the register and propagated directly to the MAC array. If the flag is 0, the switch prevention register is disabled and the previous value is kept as input to the MAC array. This way, the input to a column or row of the MAC array is kept constant, effectively lowering the switching activity. Each MAC guarding unit [Fig. 9(c)] contains 94 NAND-2 equivalent gates. The actual power reduction is dependent on the product of the sparsity levels of both the weights s_w and the feature map inputs s_i. A multiplier will only be guarded completely if both inputs are zero: $P_{\text{rel}} = s_i \times s_w$.

The same flags are used to clock gate the accumulation registers. In this case, the sparsity flags are used to clock gate whole columns or rows in the MAC array at once. This leads to much higher relative gains, as only one of the inputs of the MAC must be zero to clock gate the accumulation register: $P_{\text{rel}} = s_i + s_w \times (1 - s_i)$.

The overhead of sparsity guarding is limited due to the 2-D array topology. 32×1 b flags have to be checked in order to potentially prevent 256 MAC units from switching. In total, a 4-kB SRAM and $< 3k$ gates are used to guard 32 kB of memory and $>400k$ gates in the MAC array. Measurements of the influence of sparsity guarding on energy consumption are given in Fig. 15.

B. Compressing IO Streams for Off-Chip Communication

Off-chip communication is controlled through a DMA containing an encoder/decoder. We implement a linear compression scheme based on 2-symbol Huffman encoding. If data are zero, the 16-b data word is encoded as a $1'b0$. If the word is nonzero, it is encoded as a 17-b word $\{1'b1, 16'b\text{data}\}$. As shown in Fig. 10, this allows near-ideal linear compression with the compressed data size (C): $C = (s \times (1/n) + (1 - s) \times ((n + 1)/n))$. Here, s is the degree of sparsity and $n = 16$ is the wordlength.



(a) (b)

Fig. 10. (a) Comparison of two-symbol Huffman and random multisymbol RLE. Huffman encoding achieves near-linear compression, independent of the data distribution. In RLE, the compression rate is a function of data clustering and lower than two-symbol Huffman for random sparsity. (b) Outline of hardware implementation.

Although compression algorithms available such as run-length encoding (RLE), used in [25], could potentially achieve superlinear compression, they do not perform well on ConvNet data sets. This is illustrated in Fig. 10(a), where the plotted compression rate for RLE is a function of the data distribution and clustering, while for Huffman, it is only a function of sparsity.

An overview of the on-chip encoder/decoder is given in Fig. 10(b). All words stored in on-chip memory are decompressed, as the correct data address is crucial for pointer-based processing.

VI. Energy-Efficient Flexibility Through a Custom Instruction Set

The arithmetic blocks, memory architecture, and computational precision round and guard units are controlled in the custom processor architecture, as shown in Fig. 6. This processor is built on a baseline SIMD RISC processor with a 16-b instruction set and an 8192-word (16 kB) instruction memory using an ASIP design tool [42]. It operates a seven-stage pipeline with one fetch (IF), one decode (ID), and five execute (E1...E5) stages and is fully C-programmable using pointers. Furthermore, it is equipped with numerous standard scalar and vector ALU instructions, as well as with jump and control instructions. Hardware loop counters are built in for three nested loops.

Listing 1 illustrates the C-implementation of a typical $3 \times 3 \times C$ convolution, of which the steps are illustrated in Fig. 7. In this paper, multiple custom variable-length subinstructions are added to the instruction set to, among others, support these guard, round, and MAC operations. The ASIP's compiler can merge these subinstructions to exploit instruction-level parallelism. This is illustrated in Fig. 11, showing assembly code generated by the generated compiler of the ASIP tool for the inner loop of the C-code example in Listing 1. The listed instructions are a parallelized combination (PC_x) of basic sequential variable length subinstructions combined into one single instruction. These combined instructions are hence the parallel combination of

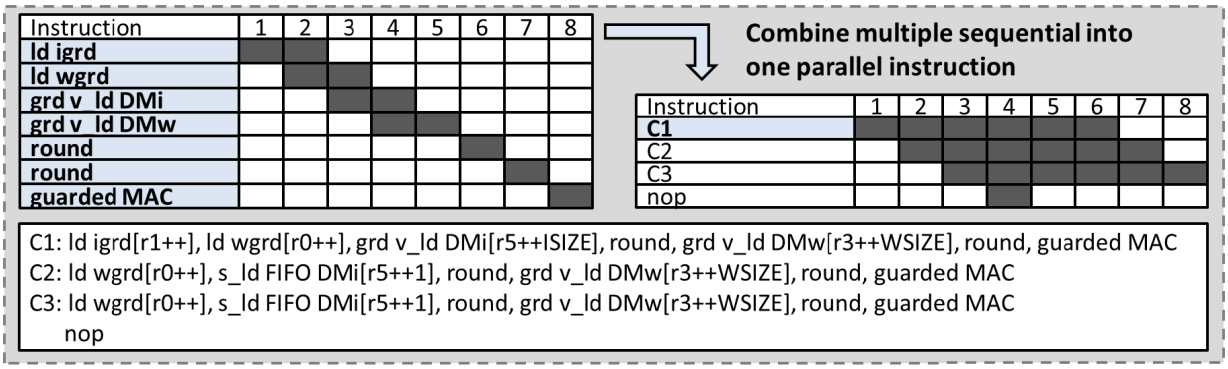


Fig. 11. Example assembly for the inner loop of Listing 1. Every line is a combination of multiple subinstructions: *ld igrd* and *ld wgrd* load the guard vectors from the guard SRAM, *s_ld FIFO DMi* and *grd v_ld DMw* perform guarded memory fetches, *round* rounds input data, and *guarded MAC* performs a guarded MAC operation. If the code is not unrolled, a nop operation is necessary inside the loop, reducing the MAC efficiency of the code.

```
for (int f = 0;f<F;f++){
 for (int h1=0;h1<H;h1++){
  for (int h2=0;h2<H;h++) {
   pW = (vint*) &Weights[f(f,h)];
    for (int c=0;c<C;c++){
     pIg = (int*) &Feature_grds[fun(f,h1,h2,c)];
     pWg = (int*) &Weight_grds[fun(f,h1,h2,c)];
     for (int k = 0; k<K;k++){
      // Fig 7. step 1, 4, 7
     | pI = (vint*) &Features[fun(f,h1,h2,c,k)];
C1   | Igr = round(guard(pI++,I,pIg));   //guard,round
     | Wgr = round(guard(pW++,W,pWg));   //guard,round
     | R = gMAC(Igr,Wgr,R,pIg++,pWg++); //guarded mac
      // Fig 7. step 2, 5, 8
     |  pIs = (int*) &Features[fun(f,h1,h2,c,k)+1];
     |  Is = fifo(pIs++);                 //FIFO ld
C2   |  Igr = round(guard(Is,I,pIg++));   //guard,round
     |  Wgr = round(guard(pW++,W,pWg++)); //guard,round
     |  R = gMAC(Igr,Wgr,R,pIg++,pWg++); //guarded mac
      // Fig 7. step 3, 6, 9
     |  pIs = (int*) &Features[fun(f,h1,h2,c,k)+2];
     |  Is = fifo(pIs++);                 //FIFO ld
C3   |  Igr = round(guard(Is,I,pIg++));   //guard,round
     |  Wgr = round(guard(pW++,W,pWg++)); //guard,round
     |  R = gMAC(Igr,Wgr,R,pIg++,pWg++); //guarded mac
     }
    }
   }
  }
 }
```

Listing 1. Example code for a 3 × 3 × C CONV layer. *pI* is a vector pointer, *Ig* is the guarded vector, and *Igr* is the guarded and rounded vector. The compiled assembly associated with this code is shown in Fig. 11. C1–3 are as in Fig. 11.

the 2 *guard fetch*, 2 *guarded vload*, 2 *round*, and *guarded MAC* instructions into one 16-b word. This code compaction should not be done manually, but is performed automatically by the ASIP's compiler, which can both call the subinstructions sequentially or in parallel, depending on the C-code. Using these parallel instructions, the number of cycles for the inner loop of Listing 1 is reduced from $3 \times 7 = 21$ to $3 + 1 = 4$, increasing the MAC efficiency and hence throughput by more than 5×. Moreover, this also drastically improves energy efficiency, as it decreases the required amount of instruction memory fetches and decoding, hence lowering the noncompute overhead, and minimizes unnecessary return to zero in the MAC array in between computations.

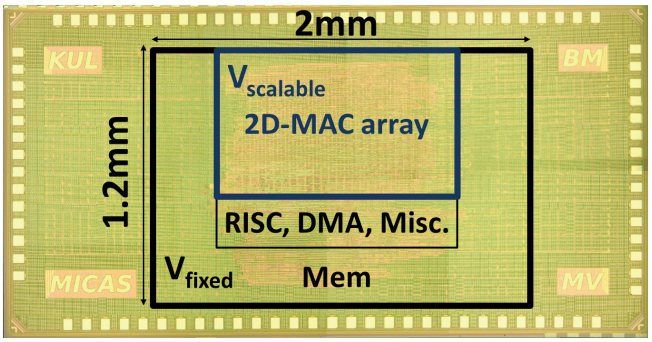


Fig. 12. Chip photograph and layout overview of the processor. The system totals 2.4 mm^2 in a 40-nm LP technology.

VII. Measurement Results

The ConvNet processor, with the layout shown in Fig. 12, was implemented in a 40-nm Low-Power (LP) technology on an active area of 2.4 mm^2. Section VII-A discusses the baseline performance. Sections VII-B and VII-C discuss the individual gains of DVAS and sparsity guarding, respectively. Section VII-D demonstrates the influence of the combined techniques on a number of benchmarks.

A. Processor Architecture Baseline

A nonunrolled $5 \times 5 \times C$ CONV layer, similar to Listing 1, is executed on the processor. At its nominal supply voltage of 1.1 V and at full precision, the processor operates at a nominal frequency of 204 MHz and a peak performance of 204 MHz × 256 MACs × 2 = 102 GOPS. Every MAC unit hence performs two operations per cycle: one multiply and one add. The real coding efficiency, the average number of MAC instructions per cycle, of the processor is typically lower and dependent on the operated filter sizes, as listed in Table V. The reference code does not achieve peak performance, but has a coding efficiency of 72%. Hence, the effective performance in this case is 74 GOPS. In this nominal mode, the processor consumes 287 mW, achieving 372 peak or 270 effective GOPS/W in energy efficiency. Table VI shows a measured power breakdown of the processor. If operated simultaneously,

TABLE V
MAC EFFICIENCY AS A FUNCTION OF FILTER SIZES

Filter Size	Not-unrolled [%]	Unrolled [%]
1×1	33	50
3×3	53	75
5×5	72	83
11×11	85	92

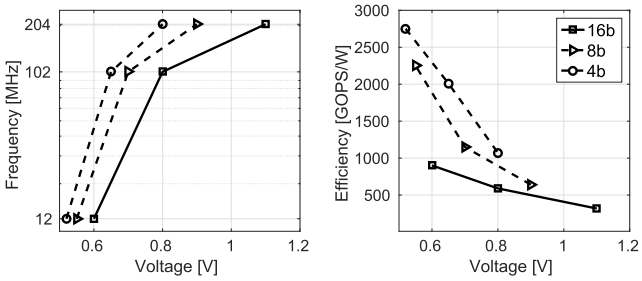


Fig. 13. (a) Frequency and (b) efficiency versus supply voltage in different modes. In the 4- and 8-b modes, the fixed power domain is running at a higher supply voltage. This voltage is the same as the one in the 16-b mode at a given frequency.

TABLE VI
FULL-PRECISION POWER BREAKDOWN AT 1.1 V AND 204 MHz

	Dynamic [mW]	Leakage [mW]	Total [mW]
Program Mem	4.1	0.4	22.5
Data Mem	18		
Control	6.4	0.3	18.7
Data Transfer	12		
MAC array	244	1.6	245.6
Total	284.5	2.3	286.8

the DMA and encoder/decoder consume an additional 2.5 mW, the extra on-chip SRAM consumption in this case ranges anywhere from 5–10 mW depending on the data sparsity and precision.

If lower throughput is allowed, the full-precision processor can easily be operated at lower voltages, as in Fig. 13. It runs at 100 MHz at 0.8 V, consuming 72 mW at 37 GOPS or 510 GOPS/W. The energy efficiency improves further up until 900 GOPS/W in the full precision mode, if operating at 12 MHz at 0.6 V, consuming only 5 mW at 4.4 GOPS.

B. Influence of Dynamic-Voltage-Accuracy-Scaling

In DVAS, the supply of the scalable power domain can be lowered at constant frequency, as illustrated in Fig. 13 for different modes. The fixed power domain is always operating at the supply necessary for full-precision operation. At 204 MHz, the scalable supply can be lowered down to 0.9 and 0.8 V for the 8- and 4-b operations, respectively. Up to 2715 GOPS/W operation is measured at 12 MHz and the 4-b precision. Fig. 14 shows the explicit effect of precision scaling and voltage scaling on the power consumption of different parts of the processor. The measured efficiency gains are significant: in a typical AlexNet l2 (Table VII), reducing precision down to 7 b, it improves the energy efficiency 2.6× compared with the full-precision baseline.

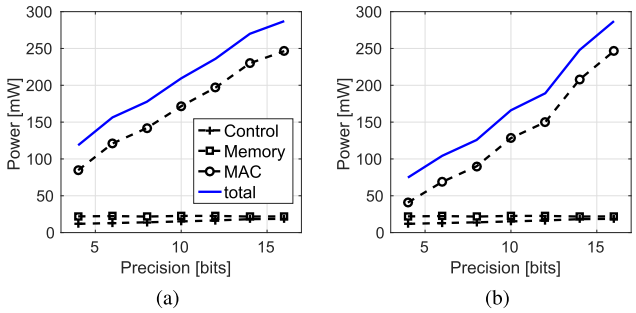


Fig. 14. Effects of precision scaling on power consumption under (a) DAS and (b) DVAS.

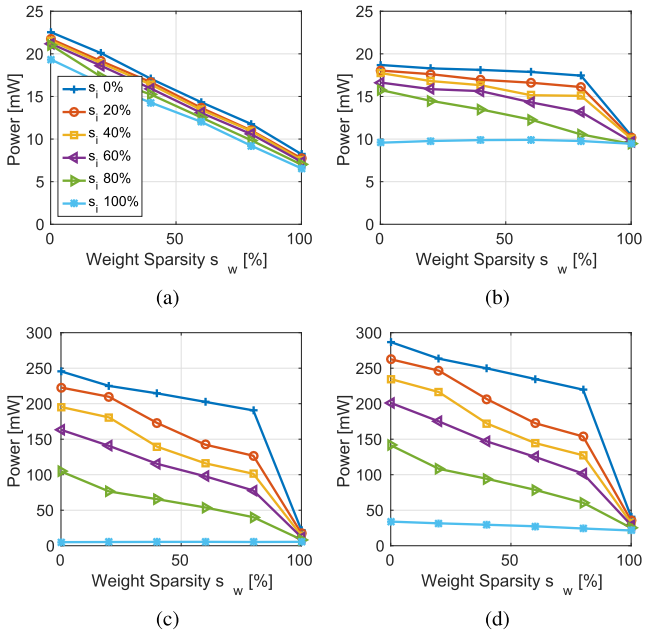


Fig. 15. Influence of input and weight sparsity guarding on the power consumption of the (a) memory, (b) control unit and data transfer, and (c) 2-D MAC array. (d) Overview of the global power consumption.

C. Influence of Sparsity Guarding

Fig. 15 shows the individual effects of sparsity guarding in different parts of the processor, using the same reference code as in the previous sections at nominal frequency and supply voltage. Memory energy consumption scales down linearly with the sparsity degrees s_w and s_i. The effect of s_i on memory power consumption is limited, because the FIFO-based architecture requires less reads from the input feature memory than from the weight memory. Energy gains are small if only one of the inputs is sparse. Still, in a typical AlexNet l3 (Table VII), relatively high sparsities $s_w = 11\%$ and $s_i = 82\%$ lead to a large 3× energy decrease in the MAC array and a 2.4× decrease for the whole system at full precision. At 100% input and weight sparsity, the dynamic power consumption is nonzero due to sparsity-flag fetch, instruction fetch, and decoding and toggling control signals.

D. Combined Effects on Benchmarks

Table VII shows the combined effect of the proposed techniques on the energy consumption of three benchmarks:

TABLE VII
PERFORMANCE COMPARISON OF RELEVANT BENCHMARKS, RUNNING AT 204 MHz

Layer	Weight bits	Input bits	Weight Sparsity (%)	Input Sparsity (%)	Weight BW Reduc.	Input BW Reduc.	IO (MB/f)	HuffIO (MB/f)	Voltage (V)	MMACs/ Frame	Power (mW)	Effective TOPS/W
General CNN	16	16	0%	0%	1.0×	1.0×	–	–	1.1	–	**287**	**0.3**
AlexNet l1	7	4	21%	29%	1.17×	1.3×	1	0.77	0.85	105	85	0.96
AlexNet l2	7	7	19%	89%	1.15×	5.8×	3.2	1.1	0.9	224	55	1.4
AlexNet l3	8	9	11%	82%	1.05×	4.1×	6.5	2.8	0.92	150	77	0.7
AlexNet l4	9	8	04%	72%	1.00×	2.9×	5.4	3.2	0.92	112	95	0.56
AlexNet l5	9	8	04%	72%	1.00×	2.9×	3.7	2.1	0.92	75	95	0.56
Total / avg.	–	–	–	–	–	–	19.8	**10**	–	–	**76**	**0.9**
LeNet-5 l1	3	1	35%	87%	1.40×	5.2×	0.003	0.001	0.7	0.3	25	1.07
LeNet-5 l2	4	6	26%	55%	1.25×	1.9×	0.050	0.042	0.8	1.6	35	1.75
Total / avg.	–	–	–	–	–	–	0.053	**0.043**	–	–	**33**	**1.6**

TABLE VIII
COMPARISON OF THIS PAPER WITH PREVIOUS PUBLISHED ConvNet IMPLEMENTATIONS

Reference	DAC'15 [21]	DAC'15 [21]	GLSVLSI'15 [29]	VLSI'16 [26]	ISSCC'16 [25]	**This Work**
Technology	E5-1620v2 CPU	Tegra K1 GPU	65nm	40nm	65nm LP	**40nm LP**
Gate Count $[NAND-2]$	-	-	912k	-	1852k	**1600k**
Core Area $[mm^2]$	-	-	1.31	1.41	12.25	**2.4**
On-chip SRAM $[kB]$	-	-	43	52 (Reg)	181.5	**148**
# c-MAC units [-]	-	-	196	2432 (eq.)	168	**256**
Nominal Frequency [MHz]	3700	852	500	240	200	**204**
Peak Performance [GOPS]	118	365	196	898	67	**102**
Average Performance [GOPS]	35	84	145	-	60	**74**
Bitwidth [bits]	32 float	32 float	12 fixed	8 fixed	16 fixed	**1-16 progr.**
Filter-sizes [-]	all	all	$< 7 \times 7$	$< 8 \times 8$	1-12[h], 1-32[v]	**all**
Channels [-]	all	all	1-256	3, 16	1-1024	**all**
Filters [-]	all	all	1-256	16, 64	1-1024	**all**
Layers [-]	all	all	all	2	all	**all**
Stride support [-]	all	all	-	no	1-12[h], 1-4[v]	**1-4 [h], all [v]**
Power-Scalability @ f_{nom} [mW]	130000	11000	510	141	235 - 332	**25 - 300**
Energy-Scalability [GOPS/W]	0.15	8.6	437-803	6400	160 - 250	**270 - 2750**
Power (AlexNet) [mW]	-	-	-	-	278	**76**
Throughput (AlexNet) [fps]	-	-	-	-	34.7	**47**

AlexNet [5], Lenet-5 [3], and the full-precision baseline reference layer of Section VII-A. All measurements are again performed at the nominal 204 MHz.

As shown in Table VII and Fig. 16, the filter weights and feature map inputs can be represented using only 7 b, reducing global power consumption by 1.9× from 274 mW at full precision down to 142 mW. The voltage in the MAC array can be set to 0.9 V, lowering power by an additional 1.3× to 107 mW. With these quantization settings, sparsities in the filters are 20% and 89% in the input feature map. If operations and memory fetches are guarded as well, the power consumption goes down another 1.9× to 55 mW, which is a total 5× gain compared with the already efficient 2-D baseline processor architecture. Also, because of sparsity, the IO communication can be compressed up to 5.8× for input features. Similar results are illustrated for the other layers in Table VII, resulting in an average of 76 mW or 0.9 TOPS/W for AlexNet, running at 47 frames/s.

LeNet-5 is more sparse and requires less computational precision, even down to 1 b in the first layer, gaining another factor of 2× in energy efficiency. The combination of the effects—lower precision arithmetic, lower voltages, and more guarded operations—allows running LeNet-5 at 13 kframes/s

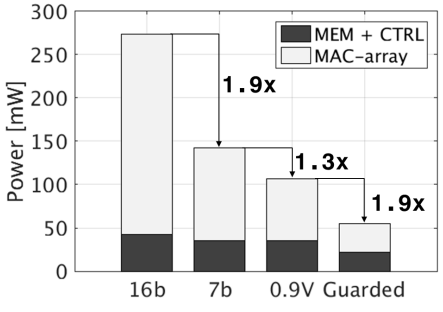


Fig. 16. Influence of the different techniques on the power consumption of AlexNet layer 2. A total 5× gain in power consumption is achieved through precision, voltage scaling, and sparse operator guarding.

at an efficiency of 1.6 effective TOPS/W or 2.5 μJ/frame. We hereby illustrate the flexibility, performance, and unique energy scalability of this design.

VIII. COMPARISON WITH THE STATE-OF-THE-ART APPROACHES

As shown in Table VIII, CPU and GPU implementations [21] are extremely flexible but consume > 100 W at low

energy efficiency. Origami [29], running at 12 b achieves an energy efficiency of 437 GOPS/W, but cannot scale its energy consumption depending on the application's requirements. Reference [26] is a hardwired ASIC for a fixed two-layer network topology, achieving high performance only when the network is $> 90\%$ sparse. Eyeriss [25], implemented in a 65-nm CMOS technology and running at 16 b, consumes 278 mW or 166 GOPS/W on the AlexNet benchmark at a throughput of 34.7 frames/s on the CONV layers or an efficiency of 8 mJ/frame.

This paper achieves a 47-frames/s throughput on the AlexNet CONV layers at nominal speed, consuming 76 mW or 1.6 mJ/frame if the system is fully optimized. This is hence a $5\times$ improvement over the AlexNet-benchmarked reference. The processor further allows scaling energy efficiency depending on the network's requirements. LeNet-5 consumes only 25–33 mW or 1600 GOPS/W at the same nominal clock frequency of 204 MHz, due to DVAS and sparsity guarding. No other work allows such network-dependent energy scalability at nominal throughput.

IX. Conclusion

A precision-scalable processor for ConvNets with operator and memory guarding was fabricated in a 40-nm LP CMOS technology. It has a total active area of 2.4 mm^2 and runs at a nominal frequency of 204 MHz at 1.1 V.

First, the processor is fully C-programmable and uses a 256 2-D MAC array architecture as an efficient convolution baseline. This 2-D architecture allows inherent $16\times$ reuse of filter weights and feature map inputs. An FIFO further reduces the internal memory bandwidth up to $27.8\times$.

Second, the processor minimizes energy by modulating precision and supply voltage in the MAC array from layer to layer. This requires extra rounding circuitry split into two power domains and an advanced backend place and route optimization. When scaling down from 16 to 8 or 4 b at 204 MHz, the scalable supply voltage can go down from 1.1 V nominally to 0.9 and 0.8 V, respectively. In AlexNet layer 2, this leads to a $2.6\times$ gain compared with the full precision baseline. Only 32 rounding units are necessary to supply rounded inputs to 256 MAC units.

Finally, sparsity in ConvNets is exploited through preventing SRAM banks and the 2-D array from switching and by compressing data up to $5.8\times$. In AlexNet layer 2, the sparsity leads to a $2\times$ gain compared with the precision- and voltage-scaled baseline, while only 32 sparsity flags are required to potentially prevent 32 SRAM banks and 256 MAC units from switching.

Because of these techniques, this chip minimizes energy consumption for any ConvNet, showing up to $5\times$ improvement over the state-of-the-art approaches in terms of energy/frame. We hereby enable low-power and high-performance applications of CV for battery-powered devices.

Acknowledgment

The authors would like to thank E. Wouters, L. Folens, and S. Redant for their backend support and S. Lauwereins, S. Smets, and H. Reyserhove for the review and discussion.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[2] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2012, pp. 4277–4280.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[4] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proc. 30th Int. Conf. Mach. Learn. (ICML)*, 2013, pp. 1058–1066.

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

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

[7] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 1–9.

[8] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit. (CVPR)*, Jun. 2014, pp. 580–587.

[10] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[11] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.

[12] J. Donahue *et al.*, "Long-term recurrent convolutional networks for visual recognition and description," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 2625–2634.

[13] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Oct. 2013, pp. 13–19.

[14] B. Moons, B. D. Brabandere, L. van Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2016, pp. 1–8.

[15] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. (Feb. 2016). "Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1." [Online]. Available: https://arxiv.org/abs/1602.02830

[16] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," *CoRR*, vol. abs/1604.03168, 2016. [Online]. Available: http://arxiv.org/abs/1604.03168

[17] W. Sung, S. Shin, and K. Hwang. (Nov. 2015). "Resiliency of deep neural networks under quantization," [Online]. Available: https://arxiv.org/abs/1511.06488

[18] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. (Feb. 2016). "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size." [Online]. Available: https://arxiv.org/abs/1602.07360

[19] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[20] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Proc. Deep Learn. Unsupervised Feature Learn. Workshop NIPS*, 2011.

[21] L. Cavigelli, M. Magno, and L. Benini, "Accelerating real-time embedded scene labeling with convolutional networks," in *Proc. 52nd Annu. Design Autom. Conf.*, Jun. 2015, Art. no. 108.

[22] A. Rahman, J. Lee, and K. Choi, "Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array," in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2016, pp. 1393–1398.

[23] N. Suda *et al.*, "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2016, pp. 16–25.

[24] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design space exploration of FPGA-based deep convolutional neural networks," in *Proc. 21st Asia South Pacific Design Autom. Conf. (ASP-DAC)*, 2016, pp. 575–580.
[25] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *Proc. IEEE Int. Solid-State Circuits Conf., Dig. Tech. Papers*, Jan. 2016, pp. 262–263.
[26] P. Knag, L. Chester, and Z. Zhang, "A 1.40mm^2 141mW 898GOPS sparse neuromorphic processor in 40nm CMOS," in *Proc. IEEE Symp. VLSI Circuits*, Jul. 2016, pp. 180–181.
[27] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2014, pp. 609–622.
[28] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2015, pp. 92–104.
[29] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A convolutional network accelerator," in *Proc. 25th Ed. Great Lakes Symp. (VLSI)*, 2015, pp. 199–204.
[30] J. Albericio, P. Judd, N. E. Jerger, T. Aamodt, T. Hetherington, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 1–13.
[31] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archi. (ISCA)*, Jun. 2016, pp. 243–254.
[32] B. Reagen *et al.*, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 267–278.
[33] B. Moons and M. Verhelst, "A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2016, pp. 1–2.
[34] V. K. Chippa, S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing: An integrated hardware approach," in *Proc. Asilomar Conf. Signals, Syst. Comput.*, Nov. 2013, pp. 111–117.
[35] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: The difficulty of learning long-term dependencies," in *Field Guide to Dynamical Recurrent Networks*. Piscataway, NJ, USA: IEEE Press, 2001.
[36] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," *CoRR*, vol. abs/1510.00149, 2016. [Online]. Available: http://arxiv.org/abs/1510.00149
[37] Y. LeCun and C. Cortes. (1998). *The Mnist Database of Handwritten Digits*. [Online]. Available: http://www.bibsonomy.org/bibtex/2935bad99fa1f65e03c25b315aa3c1032/mhwombat
[38] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2009, pp. 248–255.
[39] B. Moons and M. Verhelst, "DVAS: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2015, pp. 237–242.
[40] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1571–1580, Nov. 2000.
[41] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. Inst. Radio Eng.*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.
[42] B. Wu and M. Willems, "Rapid architectural exploration in designing application-specific processors," Synopsys, Mountain View, CA, USA, White Paper 04/15.AP.CS5529, 2015.
[43] Y. Lu *et al.*, "A 123-phase DC-DC converter-ring with fast-DVS for microprocessors," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2015, pp. 1–3.
[44] T. M. Andersen *et al.*, "A sub-ns response on-chip switched-capacitor DC-DC voltage regulator delivering 3.7W/mm2 at 90% efficiency using deep-trench capacitors in 32nm SOI CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 90–91.

Bert Moons (S'13) was born in Antwerp, Belgium, in 1991. He received the B.S. and M.S. degree in electrical engineering from Katholieke Universiteit (KU) Leuven, Leuven, Belgium, in 2011 and 2013, respectively. In 2013, he joined the ESAT-MICAS laboratories of KU Leuven as a Research Assistant funded through an individual grant from Research Foundation - Flanders (FWO, former IWT). In 2016 he was a Visiting Research Student at Stanford University in the Murmann Mixed-Signal group. Currently, he is working toward the Ph.D. degree on energy-scalable and run-time adaptable digital circuits for embedded Deep Learning applications. His focus is on enabling battery-powered and always-on neural network inference for computer vision and automatic speech recognition.

Marian Verhelst (S'01–M'09–SM'14) received the Ph.D. degree (*cum ultima laude*) from Katholieke Universiteit (KU) Leuven, Leuven, Belgium, in 2008.

She joined the Berkeley Wireless Research Center, University of California, Berkeley, CA, USA, as a Visiting Scholar, in 2005. From 2008 to 2011, she was with the Radio Integration Research Laboratory, Intel Laboratory, Hillsboro, OR, USA. She joined the MICAS Laboratories (MICro-electronics And Sensors), Electrical Engineering Department, KU Leuven, as an Assistant Professor in 2012. She is currently an SSCS Distinguished Lecturer and also a member of the Young Academy of Belgium. She has authored over 80 papers in conferences and journals. Her current research interests include self-adaptive circuits and systems, embedded machine learning, and low-power sensing and processing for the Internet-of-Things.

Dr. Verhelst is a member of both the ISSCC and DATE TPCs and of the DATE and ISSCC executive committees. She was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-II and is an Associate Editor of the IEEE JOURNAL OF SOLID-STATE CIRCUITS.