Open access • Journal Article • DOI:10.1109/JSSC.2016.2636225

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

Bert Moons, Marian Verhelst

Institutions: Katholieke Universiteit Leuven

Published on: 01 Apr 2017 - IEEE Journal of Solid-state Circuits (IEEE)

Topics: Energy consumption, Microarchitecture and Efficient energy use

An Energy-Efficient Precision-Scalable ConvNet Processor in 40nm 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 (CNN) is implemented in a 40nm CMOS technology. To minimize energy consumption while maintaining throughput, this work 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 102GOPS running at 204MHz and 1.1V. It is fully C-programmable through a custom generated compiler and consumes 25-287mW at 204MHz, scaling efficiency between 0.3-2.7 effective TOPS/W. It achieves 47fps on the convolutional layers of the AlexNet benchmark, consuming only 76mW. This system hereby outperforms the state-of-the-art up to 5× in energy efficiency.**

***Index Terms*—Deep Learning, ConvNet, Convolutional Neural Network, CNN, Approximate Computing, Voltage Scaling, DVAS, Dynamic-Voltage-Accuracy-Scaling, Processor Architecture.**

I. Introduction

DEEP Learning [1] networks, and more specifically ConvNets or Convolutional Neural Networks (CNN), 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 (ASR). 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].

Second, numerous references [14] [15] [16] [17] show ConvNets are inherently fault-tolerant and can be operated at low computational precision with limited accuracy loss. [15] even goes down to 1-bit (1b) 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, such as [25]–[32], are ASICs which focus on low-power embedded applications, aiming to achieve real-time operation at sub-Watt power consumption. They are all accelerators, reducing their flexibility in exchange for energy-efficiency. Eyeriss [25] proposes a two-dimensional spatial architecture, exploiting data-locality and network sparsity, but does not exploit reduced precision computations. [26] is an optimized architecture exploiting reduced precision, but can only perform a hardwired number of layers. DaDianNao [27] and ShiDianNao [28] exploit locality, but only achieve high performance for small neural networks. [29] uses a specific 7×7 convolutional engine, which limits flexibility, operating at constant 12b fixed-precision. [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 network layers.

This work [33], outperforms the state-of-the-art 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 two dimensional (2D) Single Instruction Multiple Data (SIMD) MAC-array.
2) Hardware support for Dynamic-Voltage-Accuracy-Scaling (DVAS), 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.
3) Hardware support for network compression and guarding sparse operations.

This paper is organized as follows. Section II discusses

Received August 08, 2016; revised ... ; accepted .... Date of publication April 04, 2017; Date of current version. This paper was approved by Guest editor ...

B. Moons and M. Verhelst are with KU Leuven, Department of Electrical Engineering ESAT-MICAS, Kasteelpark Arenberg 10, Leuven B-3001, Belgium.

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

Digital Object identifier xx.yyy/JSSC.201z.aaaaaa

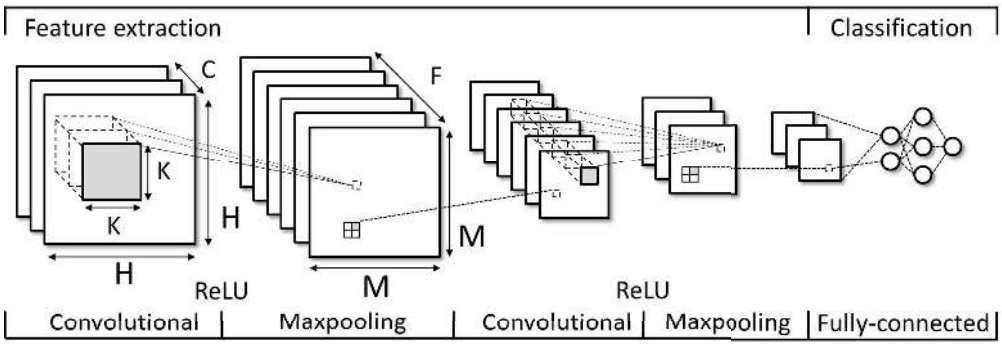


Fig. 1. Overview of a typical ConvNet architecture: 2 sequences of convolutional - ReLU - Maxpool layers perform hierarchical feature extraction. This is followed by a multi-layer fully connected classification stage.

the basics of ConvNets and the possibilities for efficiency optimization. Section III discusses the processor architecture. Section IV and section 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 work.

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-, non-linearity- and pooling-layers used for feature extraction, followed by a smaller number of fully-connected 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 3 fully connected layers [5].

A **convolutional layer** (CONV), with 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

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

TABLE II
Model-size and computational complexity comparison between 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

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, 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 **non-linearity 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 non-activated outputs.

Max-pooling 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, **fully connected layers** (FC) are used as classifiers in the ConvNet algorithm. An FC layer is mathematically described as the matrix-vector product $O[z] = \sum_{m=0}^{M} W[z, m] \times$

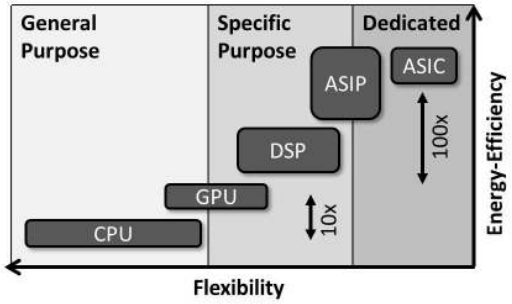


Fig. 2. Different hardware platforms in the energy-efficiency vs flexibility space. ASIP's offer better flexibility than hardwired or reconfigurable ASICs at a similar energy efficiency.

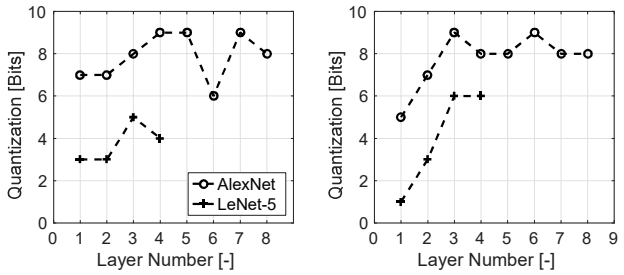


Fig. 3. Necessary number of bits for (a) weights and (b) input feature maps. With these quantization settings [14], the network achieves 99% relative accuracy compared to full precision 32b floating point operation.

$I[m]$. Where, M is the size of vectorized input feature map and $z \in [0, \ldots, Z[$ is the number of neurons in the fully connected 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 [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 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 speed-up 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-vs-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 GPU's 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 for Application-Specific Integrated Circuits (ASIC), 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 work hence proposes the design of a C-programmable Application-Specific Instruction set Processor (ASIP), completely optimized for ConvNet data flows while maintaining flexibility.

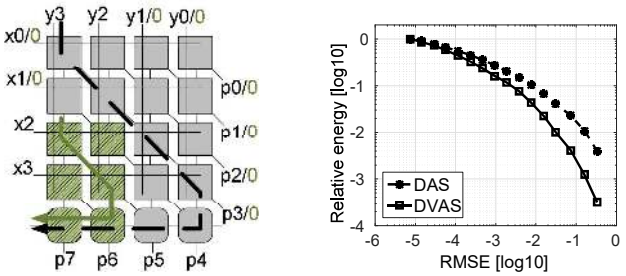
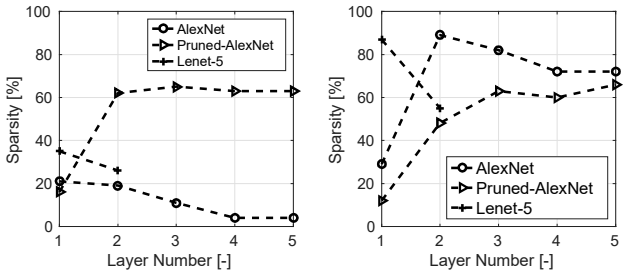


Fig. 4. (a) The basic principle of DVAS in a digital Baugh-Wooley multiplier example. (b) When precision is reduced by signal-gating less-significant bits both the switching activity (DAS) and the critical path length are reduced, allowing for a reduced supply voltage for the whole multiplier. The combination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

*2) **DVAS: Dynamic-Voltage-Accuracy-Scaling**:* Typically, ConvNets are computed using 32b Floating-Point precision [15] [16] [17]. However, they can be operated at Fixed-Point precision, at lower energy per operation. [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 1- to 6b precision suffices for LeNet-5 [3] on MNIST [37] data, and 5- to 9b for AlexNet [5] on ImageNet [38] data, at 99% relative benchmark accuracy compared to a 32b floating point baseline.

A multiplier, capable of modulating its precision and supply voltage was first proposed in [39]. The simplified example in Figure 4 can operate from 1- to 4b, with two operating modes (2b and 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 (dashed line). In the low-precision case, only the two most significant bits (MSB) are used, resulting in a lower switching activity and shorter critical path (full line). The LSB's 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



(a) Weights (b) Input Feature Maps

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, from [19].

ranging from tens of nanoseconds [**?**] to sub-nanosecond [**?**].

These combined effects - (1) reduction of switching activity and (2) shorter critical paths allowing lower supply voltages - have a major impact on the system's dynamic power consumption. If leakage power is neglected, the power P of a DVAS-system is given in equation 2.

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

Where α is the circuit's switching activity, C is the technology dependent circuit capacitance, f is the clock frequency and V 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 Dynamic-Voltage-Accuracy-Scaling (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 (DVFS) [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 input 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.

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

III. A 2D-MAC PROCESSOR ARCHITECTURE FOR EMBEDDED CONVNETS

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

A. Processor Datapath

1) ***2D MAC-array****:* The 16×16 2D-MAC array, shown in Fig. 6 and 7 operates as a convolution engine. First, the array of single-cycle MAC's achieves a $256\times$ speedup compared to a scalar solution, while minimizing bandwidth to on-chip memory. The 2D architecture allows applying 16 different filters to 16 different units of the input feature map simultaneously. This exploitation of data reuse thus allows a $256\times$ speedup, at more than $16\times$ lower internal bandwidth compared to a naive 1D-SIMD solution, requiring 2 inputs per processing unit per cycle. The local communication overhead can be further reduced by adding a 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 2D-MAC array architecture compared to the naive 1D-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 4 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 4 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, 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 48b, which is an over-design 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.

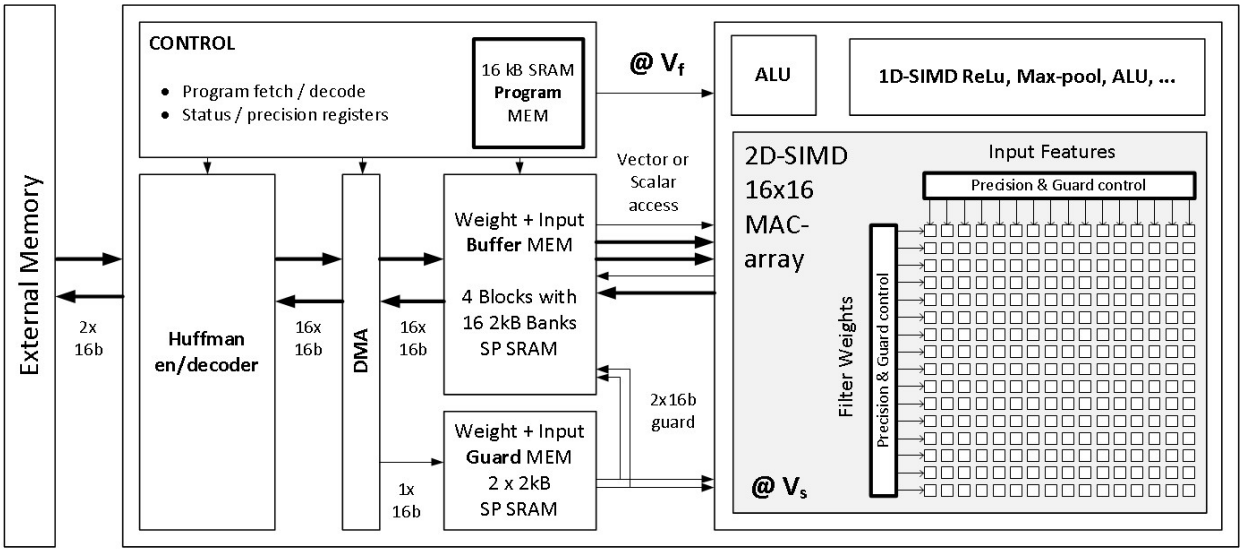


Fig. 6. High level overview of the processor. The chip contains a scalar and $16\times$ vector ALU, an input and weight on-chip SP SRAM, an SRAM storing sparsity information, a control unit, a DMA and a Huffman-based en/decoder in a fixed power domain at supply V_f. A 2D SIMD MAC array is put in a scalable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

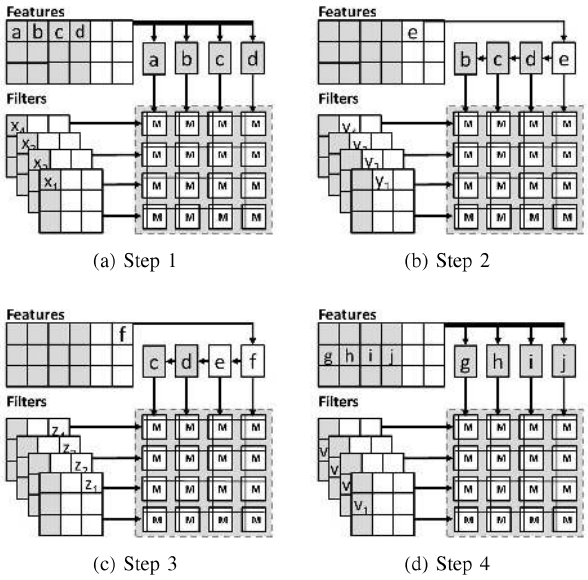


Fig. 7. Example of a typical ConvNet dataflow in the processor. In the first step, a vector of three input feature map units a, b, c is fetched and multiplied with a vector of filter weights. The results are accumulated with the intermediate result that is stored in the MAC's accumulation register. In step 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.

*2) **1D SIMD-unit**:* The processor also contains a 1D-SIMD processing unit capable of performing multiple types of vector operations. This vector unit contains 16 parallel processing units with support for bitwise- and shift operators, MAC and max-pooling (2×2 and 3×3).

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

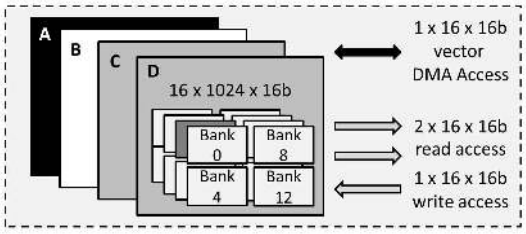
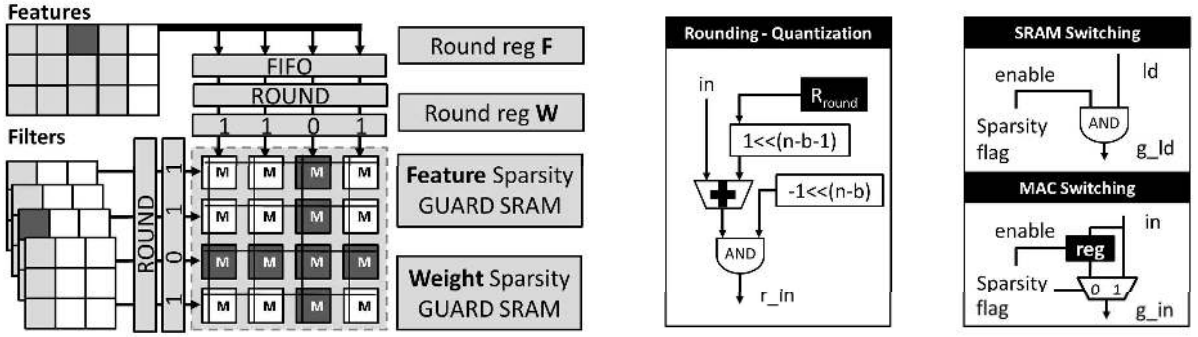


Fig. 8. On-chip memory architecture. One large address space is subdivided into 4 blocks, each containing 16 banks of single port SRAM. Every SRAM macro can store $1024 \times 16b$ words. The architecture allows scalar, vector or double vector access from the processor side, while being read or written through a vector port from the DMA side.

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 16b memory address space (128kB) and is subdivided into 4 blocks (32kB) containing 16 single port SRAM banks (2kB) of $1024 \times 16b$ 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 2D-MAC array in a single cycle. There is only one write port from the processor side, due to the lower amount of write accesses.



(a) Rounded + Guarded MAC-array

(b) Programmable rounding unit

(c) SRAM + MAC guarding

Fig. 9. The 2D-array is expanded with programmable rounding and memory fetch- and operator-guarding support. The inputs of the array (a) are rounded and guarded through arrays of (b) programmable rounding units and (c) guarding units. In (a) the guard flags are indicated: they prevent two SRAM banks and a row and 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) the sparsity flag is used as an enable signal. g_ld and g_in are respectively the guarded load or MAC-array inputs.

*2) **Direct Memory Access Controller**:* A custom designed DMA allows to efficiently communicate 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 processor and DMA is done by checking DMA-specific status registers, indicating if data transfer is done or not.

Furthermore, it contains a specific Huffman-based en/decoder [41] performing IO compression on sparse in- and output data (section V-B). Communication with the outside world is done using a 32b 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 back-end 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 Most-Significant-Bits (MSB). The Least-Significant-Bits (LSB) 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 3b MSB. This process requires two additions to the processor architecture.

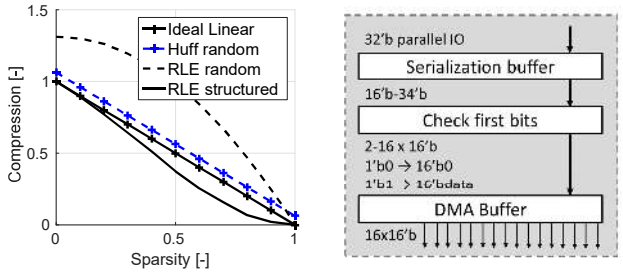
First, two programmable status registers are added, as in Fig. 9a, one for feature map inputs and one for filter weights, containing the number of bits that will be used for computation. These status registers can be written from software with a latency of 2 cycles. In practice, the precision is only 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. 9a. Fig. 9b 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 2D-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.

B. Physical implementation

Creating a multi-power 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 back-end optimization flow should be changed as well.

DVAS is not automatically possible if the back-end 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 sub-optimal placement, high wire delays or small transistor sizings. Therefore, in DVAS it is necessary to perform an advanced multi-mode, multi-corner place and route optimization scheme enforcing the theoretical potential of shorter critical paths at lower precision in the physical implementation. Ideally, layout should be optimized for the continuous precision range from 1-16 bit, using libraries characterized at all supply voltages. However, satisfactory results where achieved optimizing only for the $4-$, $8-$, $12-$ and $16b$ modes. The design was optimized for 200MHz operation in different simultaneously performed optimization modes, listed in Table IV. If libraries where not available at the necessary voltage, the circuit was optimized at the lowest available voltage with heavier timing constraints, effectively mimicking lower-voltage operation.



(a) RLE vs Huffman (b) Implementation

Fig. 10. (a) (a) Comparison of two-symbol Huffman and random, multi-symbol RLE-encoding. 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.

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. 9a), containing $1024 \times 16b$ words each ($4kB$ total), storing the sparsity information for one main memory block. Every word is a $16 \times 1b$ 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 both for guarding unnecessary memory fetches from and for guarding switching in the MAC array.

*1) **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. 9c. $16 + 16$ $1b$ flags are checked in order to potentially prevent 32 large SRAM banks from switching.

*2) **Guarding 2D-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. 9c) 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: $P_{rel} = s_i \times s_w$.

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:$P_{rel} = s_i + s_w \times (1 - s_i)$.

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

```
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 3x3xC CONV layer. *pI* is a vector pointer, *Ig* is the guarded vector, *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

B. Compressing IO streams for off-chip communication

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

TABLE IV
Multi-mode 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

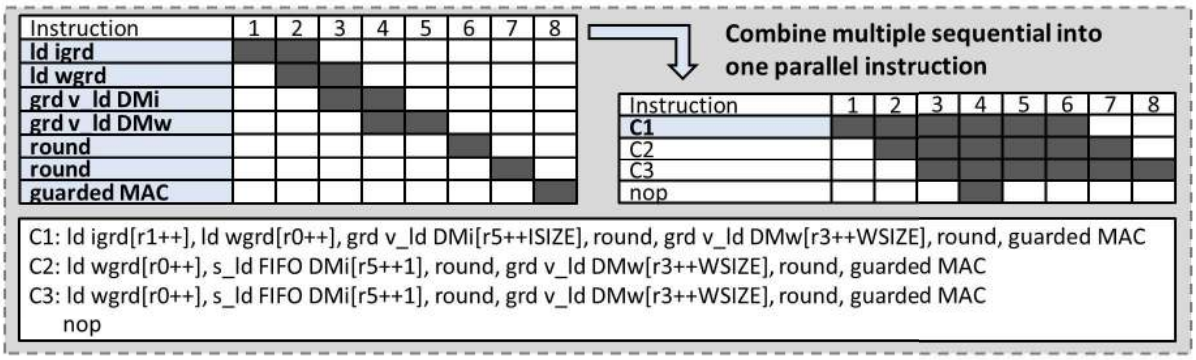


Fig. 11. Example assembly for the inner loop of listing 1. Every line is a combination of multiple sub-instructions: *ld igrd* and *ld wgrd* load the guard vectors from the guard SRAM, *s_ld FIFO DMi* and *grd v_ld DMw* perform guarded memory fetches, *round* rounds input data and *guarded MAC* performs 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.

Although compression algorithms available such as Run-Length Encoding (RLE), used in [25] e.g., could potentially achieve superlinear compression, they do not perform well on ConvNet datasets. This is illustrated in Fig. 10a, 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 en/decoder is given in Fig. 10b. 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, computational precision round- and guard units are controlled in the custom processor architecture shown in Fig. 6. This processor is built on a baseline SIMD RISC processor with a 16b instruction set and an 8192-word (16kB) instruction memory using an ASIP design tool [42]. It operates a 7-stage pipeline with one fetch (IF), one decode (ID), and 5 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 work, multiple custom variable-length sub-instructions are added to the instruction set to, among others, support these guard, round and MAC operations. The ASIP's compiler can merge these sub-instructions 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 sub-instructions combined into one single instruction. These combined instructions are hence the parallel combination of 2 *guard fetch*, 2 *guarded vload*, 2 *round* and a *guarded MAC* instruction into one 16b word. This code compaction should not be done manually, but is performed automatically by the ASIP's compiler, which can both call the sub-instructions sequentially or in parallel, depending on the C-code. By 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 $5\times$. Moreover, this also drastically improves energy efficiency, as it decreases the required amount of instruction memory fetches and decoding, hence lowering the non-compute overhead, and minimizes unnecessary return to zero in the MAC array in between computations.

VII. Measurement Results

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

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

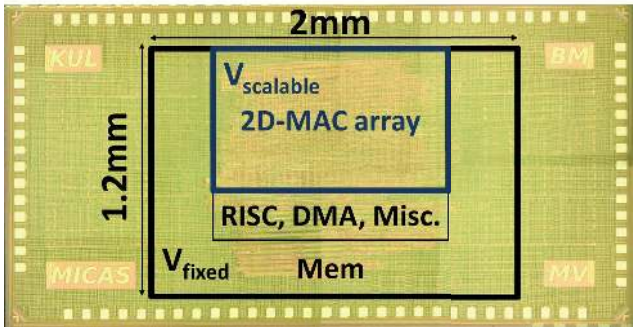
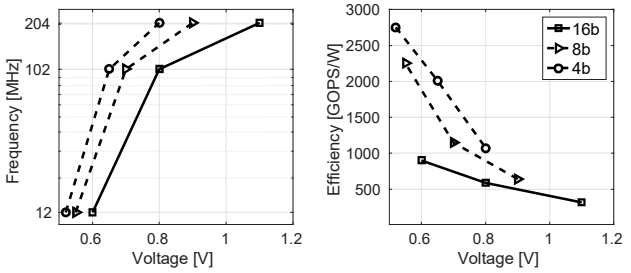
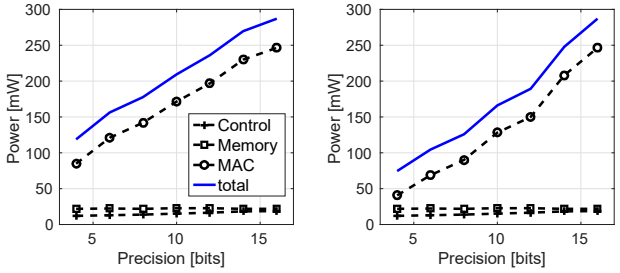


Fig. 12. Chip photograph and layout overview of the processor. The system totals $2.4mm^2$ in a 40nm LP technology.



(a) Frequency (b) Efficiency

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



(a) DAS (b) DVAS

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

A. Processor Architecture baseline

A non-unrolled $5 \times 5 \times C$ CONV layer, similar to listing 1, is executed on the processor. At it's nominal supply voltage of $1.1V$ and operating at full precision, the processor operates at a nominal frequency of 204MHz and a peak performance of 204MHz$\times$256MACs$\times$2 $=$ 102 giga-operations per second (GOPS). Every MAC-unit hence performs two operations per cycle: 1 multiply and 1 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 287mW, achieving 372 peak or 270 effective GOPS/W in energy-efficiency. Table VI shows a measured power breakdown of the processor. If operated simultaneously, the DMA and en/decoder consume an additional 2.5mW, 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 100MHz at $0.8V$, consuming 72mW at 37 GOPS or 510 GOPS/W. Energy-efficiency improves further up until 900 GOPS/W in full precision mode, if operating at 12MHz at $0.6V$, consuming only 5mW 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 204MHz, the scalable supply can be lowered down to $0.9V$ and $0.8V$ for 8b and 4b operation respectively. Up to 2715 GOPS/W operation is measured at 12MHz and 4b precision. Figure 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 7b, improves energy-efficiency 2.6$\times$ compared to the full precision baseline.

TABLE VI
FULL PRECISION POWER BREAKDOWN AT $1.1V$ AND 204MHZ.

	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

C. Influence of sparsity guarding

Figure 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 sparsity $s_w = 11\%$ and $s_i = 82\%$, leads to a large 3$\times$ energy decrease in the MAC array and a 2.4$\times$ decrease for the whole system at full precision. At 100% input and weight sparsity, the dynamic power consumption is non-zero due to sparsity-flag fetching, instruction fetch- and decoding and toggling control signals.

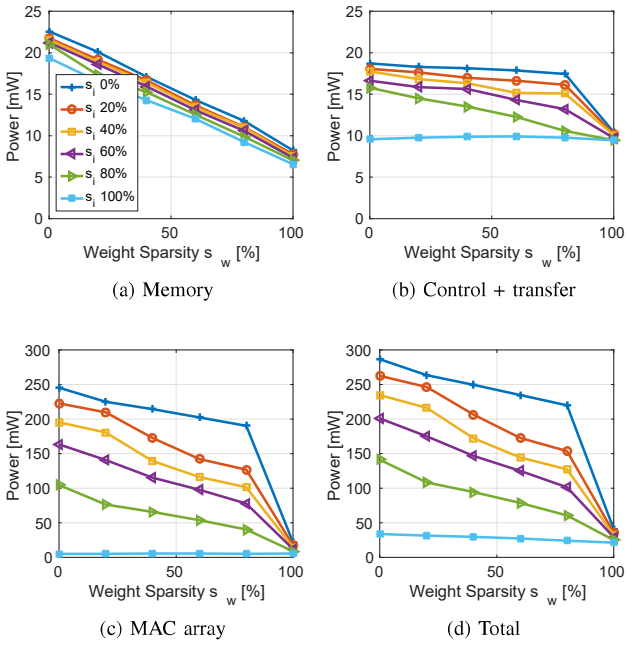
D. Combined effects on benchmarks

Table VII shows the combined effect of the proposed techniques on the energy consumption of three benchmarks: AlexNet [5], Lenet-5 [3] and the full precision baseline reference layer of section VII-A. All measurements are again performed at the nominal 204MHz.

As shown in Table VII and Fig. 16, the filter weights and feature map inputs can be represented using only $7b$. Reducing global power consumption by 1.9$\times$ from 274mW at full precision down to 142mW. Voltage in the MAC array can be set to $0.9V$, lowering power by an additional 1.3$\times$ to 107mW. With these quantization settings, sparsity in the filters is 20% and 89% in the input feature map. If operations and memory

TABLE VII
PERFORMANCE COMPARISON OF RELEVANT BENCHMARKS, RUNNING AT 204MHZ.

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**



(a) Memory (b) Control + transfer

(c) MAC array (d) Total

Fig. 15. Influence of Input and Weight sparsity guarding on the power consumption of the (a) Memory, (b) Control unit and data transfer and the 2D MAC array. (d) Gives an overview of the global power consumption.

fetches are guarded as well, power consumption goes down another 1.9× to 55mW, which is a total 5× gain compared to the already efficient 2D-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 76mW or 0.9 TOPS/W for AlexNet, running at 47 fps.

LeNet-5 is more sparse and requires less computational precision, even down to 1b in the first layer, gaining another factor of 2× in energy-efficiency. The combination of these effects - lower precision arithmetic, lower voltages and more guarded operations - allows running LeNet-5 at 13k fps at an efficiency of 1.6 effective TOPS/W or 2.5uJ/frame. We hereby illustrate the flexibility, performance and unique energy-scalability of this design.

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

As shown in table VIII, CPU and GPU implementations [21] are extremely flexible but consume $> 100W$ at low energy-effciency. Origami [29], running at 12b achieves an energy-efficiency of 437 GOPS/W, but can not scale its energy consumption depending on the application's requirements. [26] is a hardwired ASIC for a fixed two-layer network topology, only achieving high performance when the network is $> 90\%$ sparse. Eyeriss [25], implemented in a 65nm CMOS technology and running at 16b, consumes 278mW or 166 GOPS/W on the AlexNet benchmark at a throughput of 34.7 fps on the CONV layers, or an efficiency of 8mJ/frame.

This work, achieves 47fps throughput on the AlexNet CONV layers at nominal speed, consuming 76mW or 1.6mJ/frame if the system is fully optimized. This is hence a 5× 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 204MHz clock frequency, 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 40nm LP CMOS technology. It has a total active area of 2.4mm^2 and runs at a nominal frequency of 204MHz at 1.1V.

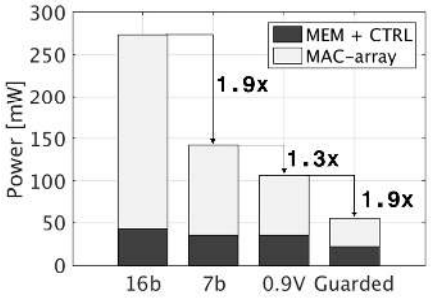


Fig. 16. Influence of the different techniques on the power consumption of AlexNet layer 2. A total 5× gain in power consumption is achieved through precision, voltage scaling and sparse operator guarding.

TABLE VIII
COMPARISON OF THIS WORK 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**

First, the processor is fully C-programmable and uses a 256 2D MAC array architecture as an efficient convolution baseline. This 2D-architecture allows inherent $16\times$ reuse of filter weights and feature map inputs. A FIFO further reduces 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, a split into two power-domains and an advanced back-end place and route optimization. When scaling down from 16- to 8- or 4b at 204MHz, the scalable supply voltage can go down from 1.1V nominally to $0.9V$ and $0.8V$ respectively. In AlexNet layer 2, this leads to a $2.6\times$ gain compared to the full precision baseline. Only 32 rounding units are necessary to supply rounded inputs to 256 MAC's.

Finally, sparsity in ConvNets is exploited through preventing SRAM-banks and the 2D array from switching and by compressing data up to $5.8\times$. In AlexNet layer 2, sparsity leads to a $2\times$ gain compared to the precision and voltage scaled baseline, while only 32 sparsity-flags are required to potentially prevent 32 SRAM banks and 256 MAC's 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 in terms of energy/frame. We hereby enable low-power, high-performance applications of computer vision for battery powered devices.

ACKNOWLEDGMENTS

The authors would like to thank the Research Foundation - Flanders (FWO, former IWT) and Intel Corporation for funding this work. We would like to thank Etienne Wouters, Luc Folens and Steven Redant for their back-end support. Steven Lauwereins, Sander Smets and Hans Reyserhove for review and discussion.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 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," *Proceedings ofIEEE international conference on Acoustics, speech and signal processing (ICASSP)*, pp. 4277–4280, 2012.

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

[4] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 1058–1066, 2013.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Proceedings of Advances in neural information processing systems*, pp. 1097–1105, 2012.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 580–587, 2014.

[10] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Proceedings of Advances in neural information processing systems*, pp. 91–99, 2015.

[11] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015.

[12] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2625–2634, 2015.

[13] M. Peemen, A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," *Proceedings of the IEEE 31st International Conference on Computer Design (ICCD)*, pp. 13–19, 2013.

[14] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1–8, 2016.

[15] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016.

[16] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," *Workshop contribution to International Conference on Learning Representations (ICLR)*, 2016.

[17] W. Sung, S. Shin, and K. Hwang, "Resiliency of deep neural networks under quantization," *CoRR*, vol. abs/1511.06488, 2015.

[18] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016.

[19] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Proceedings of Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.

[20] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Deep Learning and Unsupervised Feature Learning Workshop at Advances in neural information processing systems*, 2011.

[21] L. Cavigelli, M. Magno, and L. Benini, "Accelerating real-time embedded scene labeling with convolutional networks," *Proceedings of the 52nd Annual Design Automation Conference*, 2015.

[22] A. Rahman, J. Lee, and K. Choi, "Efficient fpga acceleration of convolutional neural networks using logical-3d compute array," *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1393–1398, 2016.

[23] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 16–25, 2016.

[24] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design space exploration of fpga-based deep convolutional neural networks," *Proceedings of the 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 575–580, 2016.

[25] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *ISSCC Dig. of Tech. papers*, pp. 262–263, 2016.

[26] P. Knag, L. Chester, and Z. Zhang, "A 1.40mm2 141mw 898gops sparse neuromorphic processor in 40nm cmos," *Proceedings of the IEEE Symposium on VLSI Circuits*, pp. 180–181, July 2016.

[27] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, 2014.

[28] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," *International Symposium on Computer Architecture (ISCA)*, pp. 92–104, 2015.

[29] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A convolutional network accelerator," *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pp. 199–204, 2015.

[30] J. Albericio, P. Judd, N. Jerger, T. Aamodt, T. Hetherington, and A. Moshovos, "Cnvlutin:ineffectual-neuron-free deep neural network computing," *International symposium on Computer Architecture (ISCA)*, 2016.

[31] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," *International symposium on Computer Architecture (ISCA)*, 2016.

[32] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernandez-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," *Proceedings of the ACM/IEEE 43rd Annual International symposium on Computer Architecture (ISCA)*, 2016.

[33] B. Moons and M. Verhelst, "A 0.3-2.6 tops/w precision-scalable processor for real-time large-scale convnets," *Proceedings of the IEEE Symposium on VLSI Circuits*, pp. 178–179, July 2016.

[34] V. K. Chippa, S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing: An integrated hardware approach," *Proceedings of the Asilomar Conference on Signals, Systems and Computers*, pp. 111–117, 2013.

[35] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[36] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *International Conference on Learning Representations (ICLR)*, 2016.

[37] Y. LeCun and C. Cortes, "The mnist database of handwritten digits," 1998.

[38] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.

[39] B. Moons and M. Verhelst, "Dvas: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing," *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 237–242, July 2015.

[40] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 11, pp. 1571–1580, Nov 2000.

[41] D. A. Huffman *et al.*, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[42] B. Wu and M. Willems, "Rapid architectural exploration in designing application-specific processors," in *ASIP Designer whitepaper*, 2015.

Bert Moons (S'13) was born in Antwerp, Belgium, in 1991. He received the B.S. ('11) and M.S. ('13) degree in electrical engineering from KU Leuven, Leuven, Belgium. In 2013, he joined the ESAT-MICAS laboratories as a research assistant after he received an individual grant from the Research Foundation - Flanders (FWO, formerly IWT). In 2016 he was a Visiting Research Student at Stanford University in the Murmann Mixed-Signal group. Currently, he is working towards a PhD 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) is an assistant professor at the MICAS laboratories (MICroelectronics And Sensors) of the Electrical Engineering Department of KU Leuven as of 2012. Her research focuses on self-adaptive circuits and systems, embedded machine learning, and low-power sensing and processing for the internet-of-things. Before that, she received a PhD from KU Leuven cum ultima laude in 2008. She was a visiting scholar at the Berkeley Wireless Research Center (BWRC) of UC Berkeley in the summer of 2005. From 2008 till 2011, she worked in the Radio Integration Research Lab of Intel Labs, Hillsboro OR. Marian is an SSCS Distinguished Lecturer, is a member of the Young Academy of Belgium, and has published over 80 papers in conferences and journals. She is a member of both the ISSCC and DATE TPCs and of the DATE and ISSCC executive committees. Marian was associate editor for TCAS-II and currently is for JSSC.