The Neuro Vector Engine: Flexibility to Improve Convolutional Net Efficiency for Wearable Vision

Maurice Peemen*, Runbin Shi†, Sohan Lal‡, Ben Juurlink‡, Bart Mesman*, and Henk Corporaal*
*Eindhoven University of Technology, the Netherlands †Soochow University, Jiangsu, P.R.China ‡TU Berlin, Germany
Email: {m.c.j.peemen, b.mesman, h.corporaal}@tue.nl shirunbin@gmail.com {sohan.lal, juurlink}@aes.tu-berlin.de

***Abstract*—Deep Convolutional Networks (ConvNets) are currently superior in benchmark performance, but the associated demands on computation and data transfer prohibit straightforward mapping on energy constrained wearable platforms. The computational burden can be overcome by dedicated hardware accelerators, but it is the sheer amount of data transfer, and level of utilization that determines the energy-efficiency of these implementations. This paper presents the Neuro Vector Engine (NVE) a SIMD accelerator for ConvNets for visual object classification, targeting portable and wearable devices. Our accelerator is very flexible due to the usage of VLIW ISA, at the cost of instruction fetch overhead. We show that this overhead is insignificant when the extra flexibility enables advanced data locality optimizations, and improves HW utilization over ConvNet vision applications. By co-optimizing accelerator architecture and algorithm loop structure, 30 Gops is achieved with a power envelope of 54 mW and only 0.26 mm^2 silicon footprint at TSMC 40 nm technology, enabling high-end visual object recognition by portable and even wearable devices.**

I. Introduction

Nowadays *machine learning* is the dominant paradigm in visual object identification and other pattern recognition tasks. Rather than imposing our real-world knowledge of objects onto static algorithms, machine learning extracts this knowledge from a rich collection of examples. Especially one category of algorithms called Deep Learning and Convolutional Networks (ConvNets) [1] have caused this shift by beating records in image recognition competitions [2], [3].

Although ConvNets achieve superior results for machine vision, it lacks an attribute crucial for mobile and wearable applications, and that is energy-efficiency. The rather large computational workload and data intensity has motivated optimized implementations on CPUs [4] and GPUs [5], but these do not fit the constrained (less than 1 Watt) mobile power budget. Our community is well aware of the trend towards heterogeneous computing where architecture specialization is used to achieve high performance at low energy [6]. A few research groups exploited this customization paradigm to design highly specialized hardware that could enable excellent machine vision for mobile devices [7]–[9].

The main challenge in accelerator design is to reconcile architecture specialization and flexibility. Especially, the right level of flexibility is key for the energy-efficiency of an accelerator. ConvNets have many parameters such as the layers, feature maps, and kernels, which are different for every task. Hence the architecture should support these different parameters efficiently, and on the other hand data storage structures should be tuned to the data-flow and data-locality requirements. Earlier works have focused on this last aspect by focusing on efficiently implementing the compute primitives. However, by adding more flexibility we demonstrate an additional efficiency improvement that is counter intuitive, but crucial for real-world ConvNet vision applications.

In this paper we present the Neuro Vector Engine (NVE), an ultra-efficient accelerator core for ConvNet vision applications. The VLIW ISA enables a very high utilization, real-world benchmarks demonstrate an excellent silicon area efficiency (performance/area = 115 Gops/mm^2). In addition, the VLIW ISA enables advanced data locality optimizations that increase energy efficiency tremendously (performance/power = 559 Gops/W). Our main contributions are:

- A new ultra-low power accelerator for ConvNets.
- Evaluate energy-consumption in the architectural design, and data movement due to inter-tile data locality optimization.
- The first extremely flexible ConvNet accelerator with full VLIW compiler support.
- Detailed comparison with a low-power Arm-A9 core and an embedded TK1 GPU.

II. Object Detection with a Convolutional Net

Convolutional Networks (ConvNets) are known as a state-of-the-art machine learning algorithm that is specialized at machine vision [1]. ConvNets are designed to process raw pixel data directly, this combines the classical model of feature extraction and classification into one algorithm. This combination is realized by simple but non-linear modules that each transform the representation at one level (starting with raw pixels) into a representation at a higher and more abstract level. In Fig. 1 this process is visualized, a series of connected 2D convolution filters build such representations. The coefficients of these filters are obtained by a learning process on a labelled dataset, which ensures that important features are extracted from the input image. For example, Layer 1 extracts simple features like edges, in further layers these are combined into more complex features such as corners and crossings. In the last layers high-level features are combined into decisions such as the detection of an object (Speed Sign) at a location in the frame, and classification of the maximum speed it represents.

A. ConvNet Structure

As outlined above a ConvNet has some differences w.r.t. classical Multilayer Perceptron Networks. The main difference

978-3-9815370-6-2/DATE16/

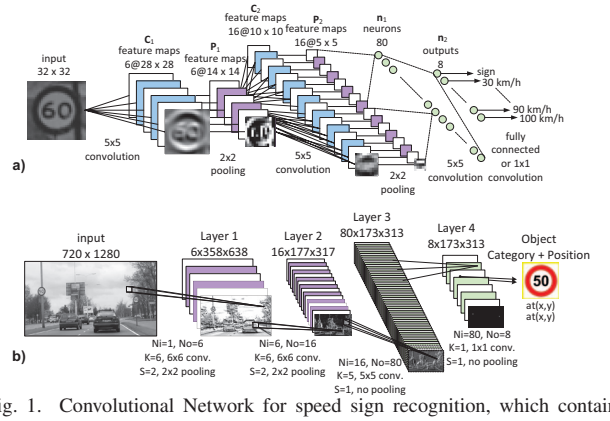

Fig. 1. Convolutional Network for speed sign recognition, which contains convolution, pooling, and classifier layers. a) Outlines the training network that is used on an image patch. b) Illustrates application version that performs real-time recognition on video frames.

is the connectivity between layers that is constrained to window operations, which are performed at multiple positions in a featuremap. In a ConvNet we observe three types of layers: convolutional, pooling, and classification.

1) Convolution Layers: The first layers in a ConvNet are often convolution layers, in Fig. 1a annotated as C_1 and C_2. Their operation is a 2D-convolution over one (C_1) or multiple input featuremaps (C_2), followed by a non-linear activation function. Listing 1 shows a code example of a convolution layer.

```
for(o=0; o<No; o++){           //output feature map
  for(m=0; m<Nm; m++){         //row in feature map
    for(n=0; n<Nn; n++){       //column in feature map
      Acc=Bias[o];
      for(i=0; i<Ni; i++)     //input feature map
        for(k=0; k<Nk; k++)   //row in convolution kernel
          for(l=0; l<Nl; l++)//column in convolution kernel
            Acc+=In[i][m+k][n+l]*Weight[o][i][k][l];
      Out[o][m][n]=sigmoid(Acc); }}}
```

Listing 1. Convolution layer loop-nest

2) Pooling Layers: To reduce the amount of data, and select only the most interesting features pooling layers are used e.g., P_1 and P_2 in figure 1a. In contrast with convolutional layers, the neighboring window operations in pooling layers have less or no overlap, which is achieved by a step size S_m, S_n. In contrast to convolution layers, pooling is done on a single input feature map. As outlined in Listing 2 pooling can be implemented by averaging or the max function. By exploiting the associativity property it is possible to combine Convolution and average Pooling into a single layer [10], which significantly reduces the workload. These combined layers are depicted in 1b, they are similar to normal convolution layers with a step size.

```
for(o=0; o<No; o++){          //output feature map
  for(m=0; m<Nm; m++){        //row in feature map
    for(n=0; n<Nn; n++){      //column in feature map
      Acc=Bias[o];
      for(k=0; k<Nk; k++)    //row in pooling kernel
        for(l=0; l<Nl; l++) //column in pooling kernel
          //subsampling version
          Acc+=In[o][Sm*m+k][Sn*n+l]*Weight[o];
          //max pooling version
          Acc=max(Acc, In[o][Sm*m+k][Sn*n+l]);
      Out[o][m][n]=Acc; }}}
```

Listing 2. Pooling layer loop-nest

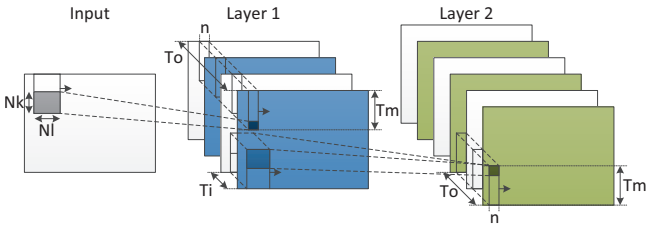

Fig. 2. Tile-strip locality optimization, maximize inter-tile reuse.

3) Classification Layers: The output of a ConvNet has one or multiple classifier layers, depicted as $n_{1\text{-}2}$ in Fig. 1a. These layers are fully-connected, which is similar as 1×1 convolution connected to all input neurons.

```
for(o=0; o<No; o++){          //output feature map
  for(m=0; m<Nm; m++){        //row in feature map
    for(n=0; n<Nn; n++){      //column in feature map
      Acc=Bias[o];
      for(i=0; i<Ni; i++) //input feature map
        Acc+=In[i][m][n]*Weight[o][i];
      Out[o][m][n]=sigmoid(Acc); }}}
```

Listing 3. classification layer loop-nest

III. ConvNet Data Locality Optimization

In recent works [9], [11] loop-interchange and loop-tiling for CNNs is introduced to improve data locality and reduce external communication. In this paper we use our tile-strip method that maximizes inter-tile reuse as demonstrated in [12]. This utilizes significantly more reuse w.r.t. classical tiling and the repetitive shifting procedure is very regular, which simplifies control. A graphical example is given in figure 2. The corresponding changes to the code due to locality optimization are outlined in listing 4.

```
for(oo=0; oo<No; oo+=To){
//tiling output feature maps
  for(mm=0; mm<Nm; mm+=Tm){
//tiling rows in feature map
    for(n=0; n<Nn; n++){          //column in feature map
      for(o=oo; o<oo+To; o++){  //output feature map
        for(m=mm; m<mm+Tm; m++){//row in featuremap
          acc=Bias[o];
          for(i=0; i<Ni; i++)   //input feature map
            for(k=0; k<Nk; k++) //row in convolution kernel
              for(l=0; l<Nl; l++)//column in convolution kernel
                acc+=in[i][m+k][n+l]*weight[o][i][k][l];
          out[o][m][n]=sigmoid(acc); }}}}}
```

Listing 4. Locality optimized loop-nest for convolution layer

To study the data transfer effects after inter-tile reuse optimization we use the exploration technique proposed in [12]. Figure 3 illustrates the resulting external transfers for a real speed sign detection application; each bar represents the transfers for a given local buffer size. A larger buffer reduces external accessess substantially, however from 8 k entries onward there is hardly any further reduction of data-transfers. Further increasing the local buffer will not result in more reuse because almost all reuse is utilized at this point. Our inter-tile reuse optimization ensures that huge amounts of reuse are utilized in a small local buffer, which is key for energy-efficiency.

IV. NVE Accelerator Architecture

The varying layer parameters in a CNN (see table II) cause that a single efficient compute operator (like in a systolic

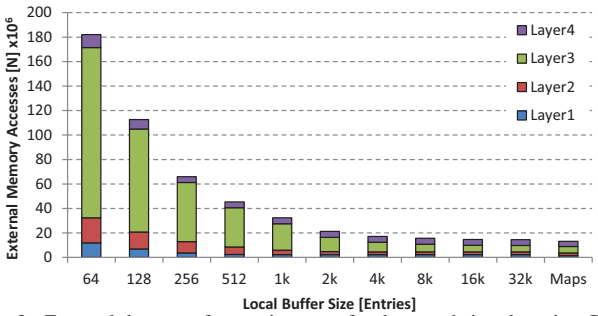

Fig. 3. External data transfer requirements for the speed sign detection CNN after inter-tile reuse optimization for different buffer sizes.

dataflow based design) is too specialized and therefore often underutilized. An efficient core for CNN based vision algorithms requires a high degree of flexibility, firstly to achieve a good hardware utilization for varying layer parameters, and secondly to implement the beneficial reordering transformations as demonstrated in section III. To obtain a good balance between efficiency and flexibility the Neuro Vector Engine (NVE) employs the Single Instruction Multiple Data (SIMD) principle i.e., we can change the operation per instruction, but share control overhead over many parallel execution units.

A. Vector Data-Path

Fig. 4 illustrates the pipelined data path of the core NVE, as the name implies all possible data operations are performed on vectors. The main compute primitive is a *Vector Multiply Accumulate* with scalar $\vec{y} \leftarrow \vec{y} + \vec{x} \times w$. It modifies an array of accumulator values $\vec{y}$ representing neighboring neurons. By sequentially repeating this operation for all inputs of a 2D convolution window this resource is well utilized for different windows sizes. Alternatively, dedicated adder trees could be used for reduction [9], [13], but reconfiguring them for varying kernel sizes is costly.

Optimized tile strips map straightforward to the array of MAC units. First a series of neighboring neurons in a feature map are initialized to a bias value, which is implemented

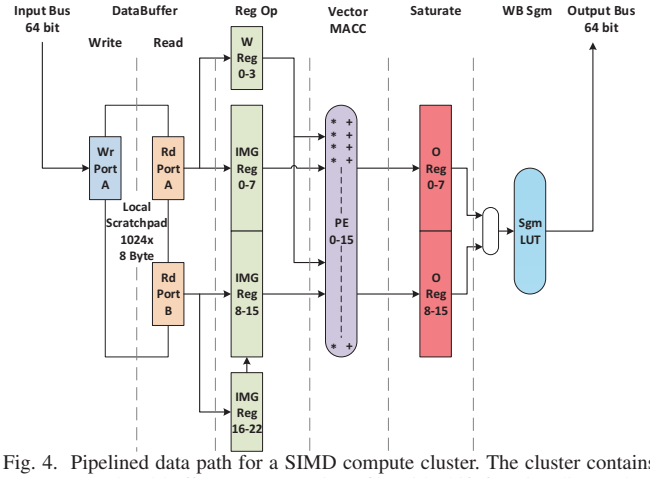

Fig. 4. Pipelined data path for a SIMD compute cluster. The cluster contains a 1024 entry local buffer, a vector register file with shift functionality, and 16 MACC units. The activation LUT has a feedback to the local buffer to enable merging of successive layers.

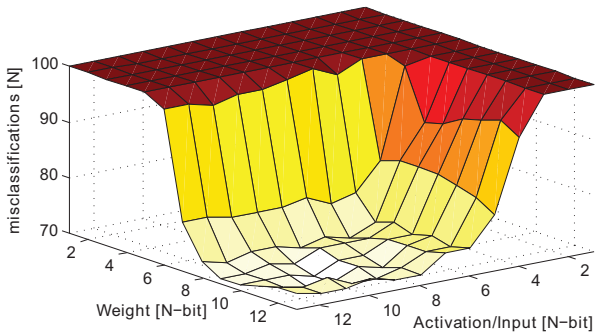

Fig. 5. Convolutional Network fixed point accuracy exploration for varying fractional precision

as setting the accumulation registers. Secondly, the PE array performs a series of MAC operations that corresponds with the kernel size. Finally if all connected inputs are accumulated the result is output to the activation function lookup tables. The operations are similar to a series of concurrent iterations of loop iterator `m` or `n`, as shown in listing 4. To achieve a high clock frequency we use a two stage pipelined MAC design.

1) Arithmetic precision: Instead of floating point data representation the more simple but energy-efficient fixed point data types are used. It is well known that neural networks do not require the range and precision of a 32-bit floating point type [14]. To obtain the minimum precision we explored the accuracy requirement of a ConvNet trained for the MNIST digit recognition task [1]. We trained the well known Lenet-5 configuration on 60,000 images using floating point precision, and evaluated the accuracy on 10,000 separate test images. The floating-point baseline has an error rate of 70 misclassifications (0.7%), which is in line with [1]. We used the MatLab fixed-point toolbox to reduce the fixed-point accuracy in the network, the results are illustrated in Fig. 5. We conclude that ConvNets require only 8-bit fractional precision, which is in line with earlier studies on classical neural nets. Table I, gives the exact configuration of the data types that we used in our accelerator design. Hence, the MACC PE array implements a 16-bit (weight) by 8-bit (input) truncated multiplication. The accumulation register is extended to 9 integer bits with a hardware check to prevent overflow. Before activation function lookup the accumulation value is saturated to the 10-bit potential format given in table I. The activation function is implemented with a look-up memory of 1024 entries of 8-bit.

B. Memory System

To exploit the data reuse in ConvNets we use efficient on-chip scratchpad buffers and specialized registers; there is no need for expensive caches because access functions are static.

1) L0 Special registers: The vector MAC unit is supplied with coefficients and input values by special vector registers,

TABLE I
FIXED-POINT DATA WORD CONFIGURATION

Data type	Word size	Fixed point format
Weights	16 bit	$SIIIIIII.FFFFFFFF$
In/Act	8 bit	$.FFFFFFFF$
Potential	10 bit	$SIII.FFFFFF$

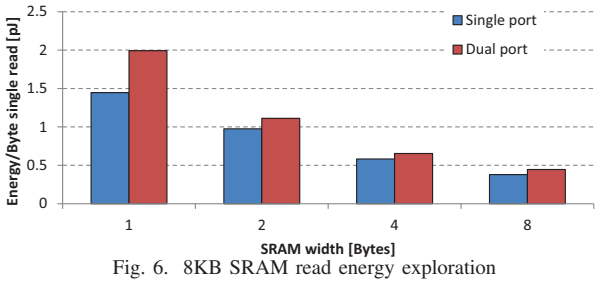

Fig. 6. 8KB SRAM read energy exploration

which are located in the Reg Op stage in Fig. 4. These registers exploit data reuse over neighboring neurons by a 1D shift and a broadcast operations. To support filter kernels of varying size a dedicated shift-in register is used, that loads vectors without waiting cycles.

2) L1 Local scratchpad: Registers load from an on-chip SRAM scratchpad to exploit the other data reuse dimensions. Combined with registers it simulates a 2D or 3D shift register. For energy-efficiency scratchpad access are 64-bit vectors, which is motivated by CACTI [15] simulation of 8kB SRAM in 40nm technology. Fig. 6 shows that 64-bit accesses are very energy efficient compared to smaller loads. Furthermore, dual-ported memory is used to improve flexibility, because multiple data streams are accessing the scratchpad, e.g., new data writes and multiple register reads can be performed concurrently at an energy overhead of only 17%.

C. Control and Programming

The control of the successive stages is distributed over *issue slots*, implemented as a Very Long Instruction Word (VLIW) core. Using instructions improves flexibility substantially, but it causes instruction fetch overhead. However, each slot is very specific, so the instruction width is small, additionally we apply: **1) SIMD:** Operate on vectors, which increases the amount of useful work per instruction; **2) Software Pipelining:** We create a schedule of instructions (Steady-State) that is repeated very often in a hardware loop buffer. In Fig. 7, an educational example of a *tile-strip* implementation of 3x3 convolution is depicted. Note that one steady-state iteration (10 cycles) processes 16 convolutions (144 MAC ops).

V. Experimental Evaluation

To evaluate the accelerator design and the loop transformation techniques we used a set of tools and applications. The settings of our setup are further outlined in the next section.

Write Local	Read Local	Reg Operation	EX VMAC	WB Sgm
	Rd a 0, [b0 w0-1]	Set i3, Shift W	MAC,w6, c2 i0	
	Rd ab 4 5, [c0,i0-15]	Shift i0 i1, Shift W	MAC,w7, c2 i1	
Wr[c3,i0-7], 4	Rd b 6, [c0,i16-17]	Set W0	MAC,w8, c2 i2	
	Rd a 1, [w2-4]	Set i0 i1, Shift W	Set, b0	
	Rd ab 7 8, [c1,i0-15]	Set i3, Shift i0 i1, Shift W	MAC,w0, c0 i0	
Wr[c3,i8-15], 5	Rd b 9, [c1,i16-17]	Shift i0 i1, Set W1	MAC,w1, c0 i1	
	Rd a 2, [w5-8]	Set i0 i1, Shift W	MAC,w2, c0 i2	Sigm0, Wr
	Rd ab 10 11, [c2,i0-15]	Set i3, Shift i0 i1, Shift W	MAC,w3, c1 i0	Sigm1, Wr
Wr[c3,i16-17], 6	Rd b 12, [c2,i16-17]	Shift i0 i1, Set W2	MAC,w4, c1 i1	
	NOP	Set i0 i1, Shift W	MAC,w5, c1 i2	

Fig. 7. Assembly description of a steady-state program for 3x3 convolution in a feature map. Note that control is distributed over the successive stages in the architecture pipeline. Image read and write actions with the local buffer use modulo addressing over steady-state iterations to maximize the efficiency of the memory.

TABLE II
Benchmark different CNN characteristics

Face Detection [16]	N_o	N_i	N_m	N_n	N_k	N_l	S
Layer 1	4	1	638	358	6	6	2
Layer 2	14	2	317	177	4	4	2
Layer 3	14	1	312	172	6	6	1
Layer 4	1	14	312	172	1	1	1
Speed Sign [17]	N_o	N_i	N_m	N_n	N_k	N_l	S
Layer 1	6	1	638	358	6	6	2
Layer 2	16	3	317	177	6	6	2
Layer 3	80	40	313	173	5	5	1
Layer 4	8	80	312	172	1	1	1

A. Benchmark Setup

The proposed accelerator is evaluated with two real-world ConvNet vision applications on 720p video, Face Detection [16], and Speed Sign Recognition [17]. These benchmarks contain a mix of different layers parameters with dense and sparse connectivity, as outlined in Table II (see Fig. 1 for interpretation). The source code of both applications is available online: http://parse.ele.tue.nl/research/NVE/

1) Baseline commercial platform mappings: As baseline for our accelerator we selected an ARM Cortex-A9 core. Due to its good energy efficiency and performance it is a popular platform for Smartphones. In addition, we compare with an Nvidia Embedded GPU the Jetson TK1 with 192 Cuda cores [18].

For fair comparison it is important that the two CNN based vision applications are optimized. For the ARM-A9, we used all compiler optimizations of GCC 4.7, which made the C implementation of speed sign detection 13.4x faster with an energy efficiency increase of 13.3x. In addition, usage of SIMD NEON intrinsics resulted in an extra speedup of 2.7x with an energy efficiency increase of 1.8x. For the TK1 mapping we used a CUDA optimized mapping, e.g., we use the constant memory for coefficients, exploit tiling and memory access coalescing.

For the ARM core we use a GEM5 [19] and McPAT [20] simulator setup. The model assumes a 4-issue superscalar core with 800MHz clock, 32kB instruction and 64kB data cache with an associativity of 2, and a line size of 64 bytes. McPAT is configured for low power operating mode in 40nm technology. The Jetson TK1 measurements are done on the NVidia development board.

2) Accelerator implementation: For evaluation of the accelerator a 16 MAC PE datapath with all control is designed in VHDL. The required memories in the design are configured as follows: an 8 kB dual-port scratchpad with 64-bit entries, eight activation function LUT memories of 1 kB each, and a 54-bit wide 512 entry instruction buffer. Although we target ASIC technology the functional correctness of the design is verified by FPGA mapping. For estimation of ASIC properties the design is synthesised with Cadence Encounter RTL Compiler with a 40 nm TSMC low-power library. The SRAM memories are simulated with CACTI [15]. The target clock frequency of 1 GHz is achieved by pipelining the stages in the datapath. It

TABLE III
BREAKDOWN OF THE AREA AND POWER OF THE NVE (40 NM)

Module	Area [mm^2]	(%)	Power [mW]	(%)
Accelerator	0.259		54.1	
Logic	0.076	(29%)	43.3	(80%)
Scratchpad	0.112	(43%)	5.9	(11%)
Act. LUT	0.045	(17%)	1.5	(3%)
Instr. Buf.	0.026	(10%)	3.4	(6%)

is important to enable retiming, since it reduced area by 32% and power consumption by 41%.

For accelerator mapping an optimizing compiler is used [21]. This compiler reads an `XML` ConvNet descriptions that is automatically converted into optimized software pipelined VLIW programmes. These programmes are automatically converted into VHDL testbenches that are simulated on the post-synthesis result for energy estimation.

B. Accelerator Characteristics

The total area footprint of the accelerator is 0.26 mm^2, which makes it a small design, e.g., the McPAT ARM core takes 14mm^2. The area breakdown of the different components in the synthesised accelerator is given in Table III. From the results it is observed that (71%) of the area is spend on the on-chip SRAM. Given the fact that the complete architecture aims at minimizing external communication by reusing data in local buffers this is in line with our expectation.

The power results are obtained by post-synthesis simulation of tile strips from the vision applications implemented as steady-state programs similar to the 3x3 convolution code in figure 7. Table III presents the average power breakdown of the simulations. The major energy portion is consumed by the logic (80%), which contains the multipliers, adders, registers, instruction decoder, etc. The on-chip memory is responsible for only a small amount of power. The main reason is the relatively small number of accesses to the memories due to data reuse in the special purpose registers, which significantly reduces the memory pressure. For example in layer 1 of the speed sign application there are 1.66 scratchpad accesses every cycle on average. However, every cycle also 16 MAC operations are performed. This large ratio explains the difference between logic power and on-chip memory accesses. Note that the instruction buffer consumes only 6% of the total power budget.

1) Throughput: With a full utilization of the vector MAC stage the accelerator theoretically performs 32 fixed point operations/cycle. Additionally, the activation function array can perform 8 activation function lookups in parallel. Combined a peak performance of 40 GOp/s can be reached at 1GHz operation. For real-world CNNs the performance number is less, time is required to load instructions, execute the prolog part, or stall cycles due to conflicts. The mapping of the layers of the speed sign detection CNN demonstrated a weighted mean of 30.2 GOp/s. Due to the high degree of register locality convolution layers without subsampling perform best in the range of 32 GOp/s. On the other hand, output layers with 1x1 convolution have almost no register locality have the worst performance 10 GOp/s.

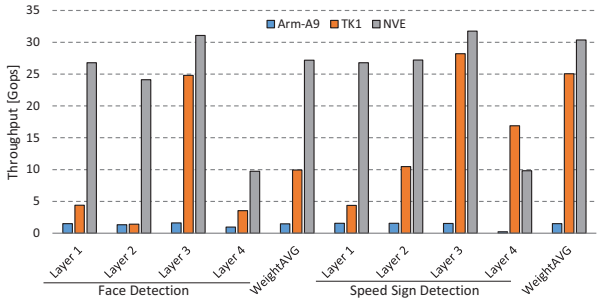

Fig. 8. Throughput comparison Arm-A9, NVidia Jetson TK1, and NVE

The throughput comparison between ARM-A9, NVIDIA TK, and the NVE is given in figure 8. The NVE shows a speedup of 20x compared to the ARM, surprisingly the NVE is also able to beat the TK1 by a factor 1.2-2.6 depending on the application. Note that the flexible NVE achieves a very constant utilization over the different layers, only the classifier layers with substantially less data reuse perform worse.

2) Energy: The energy consumption of the ARM core versus the accelerator is compared in figure 9, which shows a very similar trend. On average the accelerator reduces energy consumption by 430x, which is a lot considering we compare with an embedded ARM core that uses energy efficient SIMD operations. With a power requirement of 54 mW, a compute efficiency of $\frac{30.2\,\text{Gop/s}}{54\,\text{mW}} = 559$ GOps/Watt is achieved. As a result the average energy per fixed point operation is 1.8 pJ, which is very competitive for 40nm technology. From this observation we conclude that the accelerator does not sacrifice compute efficiency as a tradeoff for the increased flexibility.

The NVIDIA TK1 consumes 6.9 Watt of power [18] this is substantially more than the NVE with 54mW. This is expected because the TK1 is a far mor generic platform designed for Tablets with a larger energy budget.

In the energy simulations the external data transfer is not taken into account. When we add the energy spend by DMA and external memory requests, the majority of the energy budget is on memory transfers. This is not caused by a bad data-reuse strategy, but the energy consumed by the NVE is reduced to a very low level, which was the main goal of this work.

VI. RELATED WORK

The energy constraints on mobile companion devices, forces researchers to find solutions that improve efficiency. Accel-

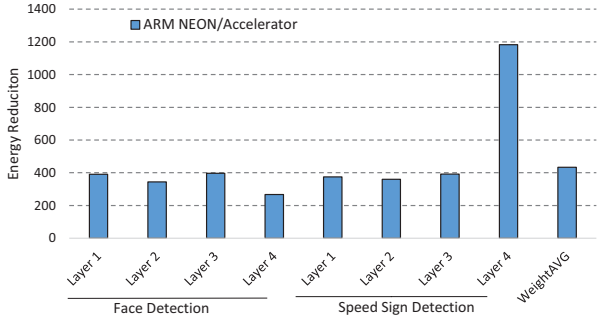

Fig. 9. Energy consumption reduction NVE vs NEON optimized ARM-A9

eration of ConvNets for FPGA or ASIC has been done by others as well. They focused on the filtering part, because that represents 90% of the computational workload. Recently, two Systolic design proposals were built that implement a 2D-convolution operation [7], [8]. Although systolic implementations are computationally efficient they have no flexibility. Therefore these proposals resort to complex arbitration logic to share inputs. Furthermore if a network does not match well with the filters, the utilization is very low. Another practical problem is the programming of these accelerators, often the programming model is neglected for simplicity.

Very recently, researchers have admitted that data transfer is an important problem for convolutional network accelerators [9], [11]. Similar to our approach they use tiling and interchange to improve locality. Although data transfer is significantly reduced, it is still the main performance bottleneck [11], or the dominating power usage [9]. We use a very flexible per instruction controllable architecture that supports advanced locality optimizations as inter-tile reuse optimization, which minimizes external data transfer. Furthermore, our NVE is designed with flexibility and efficiency in mind; this enables us to generate optimized VLIW code from a high-level `XML` network description.

VII. Limitations and Future Directions

We focused on an energy-efficient and wearable use case in the 100 mW power budget. To meet these targets we did not consider Wide vector processing, e.g. 32, 64, and 128 MAC units. Our current implementation achieves 13 fps real-time performance for speed sign detection, which is currently enough. Wide vector implementations certainly improve performance, so more design space exploration is required in our future work.

VIII. Conclusion

Customized hardware acceleration for visual object classification has significantly improved the computational efficiency of Convolutional Neural Networks (CNNs), that are currently the dominant paradigm. The current works are very specialized, which is often suboptimal for varying network configurations. We have presented a new SIMD architecture the Neuro Vector Engine (NVE) for hardware-acceleration and corresponding mapping optimizations for inter-tile reuse that exploit data locality.

A VLIW-type controller allows for sufficient flexibility to encode the CNN based machine vision applications. Synthesis (40nm CMOS) results in a pipelined circuit that operates at 1GHz, yielding an effective fixed performance of 30.2 GOp/s (20x faster w.r.t. a SIMD optimized ARM-A9). Considering the power budget of only 54 mW, this accelerator is suitable for embedment in next-generation mobile devices and bring smart features like real-time visual object classification and speech recognition to our cherished portable companions.

References

[1] 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.

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

[3] D. CireşAn, U. Meier, J. Masci, and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification," *Neural Networks*, vol. 32, pp. 333–338, 2012.

[4] K. Chellapilla, S. Puri, P. Simard *et al.*, "High performance convolutional neural networks for document processing," in *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.

[5] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, ser. IJCAI'11. AAAI Press, 2011, pp. 1237–1242.

[6] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *ISCA'37*, vol. 38, no. 3. ACM, 2010, pp. 37–47.

[7] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 247–257.

[8] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision," in *Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE Computer Society Conference on*, 2011, pp. 109–116.

[9] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*. ACM, 2014, pp. 269–284.

[10] M. Peemen, B. Mesman, and H. Corporaal, "Efficiency optimization of trainable feature extractors for a consumer platform," in *ACIVS'11*, 2011, pp. 293–304.

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

[12] M. Peemen, B. Mesman, and H. Corporaal, "Inter-tile reuse optimization applied to bandwidth constrained embedded accelerators," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15, 2015.

[13] F. Conti and L. Benini, "A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15, 2015, pp. 683–688.

[14] J. L. Holi and J. N. Hwang, "Finite precision error analysis of neural network hardware implementations," *IEEE Trans. Comput.*, vol. 42, no. 3, pp. 281–290, Mar. 1993.

[15] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to understand large caches," HP Labs, Tech. Rep., 2009.

[16] C. Garcia and M. Delakis, "Convolutional face finder: a neural architecture for fast and robust face detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 11, pp. 1408–1423, Nov 2004.

[17] M. Peemen, B. Mesman, and H. Corporaal, "Speed sign detection and recognition by convolutional neural networks," in *Proceedings of the 8th International Automotive Congress*, 2011, pp. 162–170.

[18] *NVIDIA Jetson TK1 Development Kit, Technical Brief, Bringing GPU-accelerated computing to Embedded Systems*, 2014.

[19] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[20] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. ACM, 2009, pp. 469–480.

[21] M. Peemen, W. Pramadi, B. Mesman, and H. Corporaal, "Vliw code generation for a convolutional network accelerator," in *Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '15, 2015, pp. 117–120.