Exploiting Subword Permutations to Maximize CNN Compute Performance and Efficiency

Michael Beyer*†, Sven Gesper‡, Andre Guntoro*, Guillermo Payá-Vayá‡, Holger Blume†
*Bosch Corporate Research, Robert Bosch GmbH, Renningen, Germany
†Institute of Microelectronic Systems, Leibniz University Hannover, Germany
‡Chair for Chip Design for Embedded Computing, Technical University Braunschweig, Germany
Email: michael.beyer2@de.bosch.com

***Abstract*—Neural networks (NNs) are quantized to decrease their computational demands and reduce their memory footprint. However, specialized hardware is required that supports computations with low bit widths to take advantage of such optimizations. In this work, we propose permutations on subword level that build on top of multi-bit-width multiply-accumulate operations to effectively support low bit width computations of quantized NNs. By applying this technique, we extend the data reuse and further improve compute performance for convolution operations compared to simple vectorization using SIMD (single-instruction-multiple-data). We perform a design space exploration using a cycle accurate simulation with MobileNet and VGG16 on a vector-based processor. The results show a speedup of up to $3.7\times$ and a reduction of up to $1.9\times$ for required data transfers. Additionally, the control overhead for orchestrating the computation is decreased by up to $3.9\times$.**

***Index Terms*—Application-Specific Processor, Subword Permutation, Neural Network Hardware, CNN**

I. Introduction

Neural networks (NNs) have shown to achieve exceptional performance in complex perception tasks such as object detection and classification. Current state-of-the-art NN architectures exhibit deep and complex network topologies. Their substantial computational requirements pose a challenge for mobile and embedded applications with limited compute resources, memory, and energy budget. Designing efficient, fast, but flexible NN accelerators is challenging, especially for deep NNs with diminishing feature map sizes. This becomes particularly clear for HW/mobile-friendly NN architectures such as MobileNet [1]. Here, data reuse is limited due to pointwise convolutions that use small 1×1 kernels, and depthwise convolutions that operate only in a single channel. Reusing data, however, is essential for high energy efficiency [2].

Pre- and post-processing of data is also important along with inferring NNs. While using dedicated HW accelerators for each task offers performance and power consumption advantages, application-specific processors are a better alternative due to the high diversity of NN architectures and the limited area budget of edge devices. Application-specific processors are programmable and allow to strike a balance between performance and flexibility. Therefore, they are also suitable for accelerating future applications.

To cope with requirements of different pre- and post-processing workloads such as FFTs (Fast Fourier Transform), SRP (Steered Response Power) [3], and Radar angle estimation [4], we design a vector processor with a native data width of 16-bit [5]. Compute performance for the mentioned applications can easily be increased by leveraging their high data-parallelism. Here, vertical vector processing allows for the required flexibility to accelerate different workloads [6]. Compared to horizontal vector processing (i.e., SIMD, single-instruction-multiple-data), vertical vector processing addresses vector elements sequentially. Consequently, various algorithms can be mapped efficiently on HW by adapting the addressing scheme (e.g., linear, strided, or other complex addressing modes [5], [7]). The large native data width of 16-bit, however, is not efficient for inferring NNs that can be quantized to 8-bit or even lower [8], [9].

In this work, we tackle the challenge of accelerating NNs on application-specific vector processors. We implement a multi-bit-width arithmetic unit to increase the compute performance and subword permutations to enhance the data reuse, which improves compute utilization and contributes to lower energy consumption. Our approach is motivated by the computational flow of convolutions and the associated semantic relationship of the processed data, instead of just improving compute performance using SIMD. In our architecture, we consider the channel-to-subword assignment during processing which allows us to effectively reuse data and thus reduce redundant loading of data. Furthermore, our subword permutation technique allows us to enhance support for mobile-friendly NN architectures which is not addressed in related work [10], [11]. We show that even when using uniform quantization levels across all layers (e.g., 8-bit), performance and efficiency of NN processing can be significantly increased.

To summarize, our contributions are:

- A MAC (multiply-accumulate) unit design supporting computations with different bit widths to increase the raw compute performance for NN workloads. Further, we include a subword permutation stage for one operand. We show how this feature enables improvements for multiple criteria during NN processing (see Sec. II and III-B). Our design is not limited to vector processors and can be extended to other accelerator architectures.
- Using extensive cycle accurate simulations, we conduct a design space exploration on a scalable vector processor. We evaluate our proposed design using the two NNs

DOI 10.1109/ASAP57973.2023.00023

MobileNet and VGG16. The results confirm that required clock cycles, data transfers, and instructions can be reduced significantly.

This document is structured as follows. In Sec. II, we provide details on the benefits of our subword permutation technique and present our proposed MAC unit architecture. Additionally, we show how convolutions are performed using our proposed design. The experimental setup and the results are presented and discussed in Sec. III. Finally, we compare our method with related work in Sec. IV and conclude our paper in Sec. V.

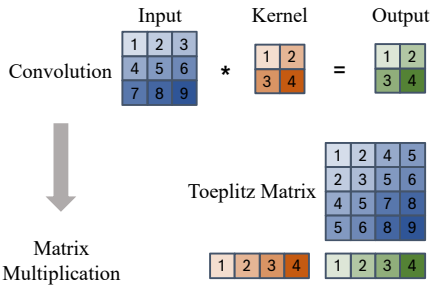
II. Method

A. Problem Statement

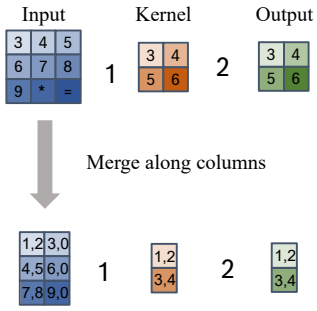
Since pre- and post-processing workloads are processed on the same HW, adding support for computations with different bit widths allows to fully utilize available computational resources during NN processing. To this end, we combine the flexibility of vertical vector processing with horizontal vector processing (i.e., SIMD) to achieve additional performance improvements. Convolutional layers are the basic ingredient of convolutional NNs. Instead of sliding a kernel over the input, NN accelerators [2], [10], [11], [13], [14], CPUs, and GPUs usually compute convolution layers by transforming the convolution operation into a matrix-matrix multiplication [12]. Fig. 1a shows how the *im2col* algorithm [15] is used to construct a Toeplitz matrix for the input feature map of a layer. Considering a 2D convolution with a $k \times k$ kernel, the redundant data in this matrix causes a k^2 larger input feature map [16]. However, the resulting matrix-matrix multiplication can be efficiently computed by leveraging optimized software libraries for General Matrix Multiplication (GEMM) [17].

The increased memory footprint of the im2col approach can be challenging for edge devices with a limited memory budget. Using the flexible addressing offered by application-specific processors enables regular processing of convolutions by sliding kernels over the input feature map. Consequently, additional transformation operations and their associated memory overhead can be circumvented. In Fig. 1b, a vectorization approach is shown, where individual elements of kernels and input feature maps are merged horizontally in subwords. This is not ideal, as sliding a kernel over the input using this approach involves memory accesses on subword level. Furthermore, depending on the kernel and feature map sizes, memory usage is not optimal. Matrices with an odd number of columns result in empty subwords as shown for the input feature map in Fig. 1b. Here, the 3×3 feature map still occupies 3×2 full width data words and three subwords are not used.

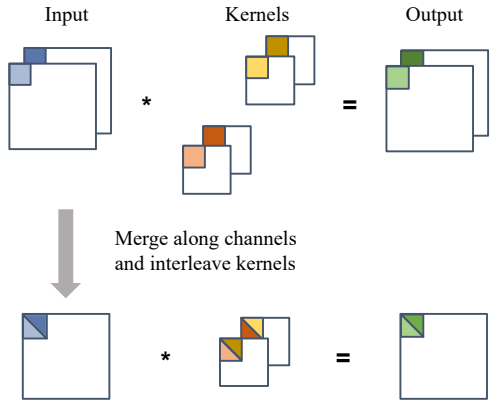
Our approach takes into account the computational flow of convolutions as well as the semantic relationship/context of the processed data, rather than merely improving performance using SIMD. As illustrated in Fig. 1c, we merge channels of input and output feature maps in subwords. Corresponding kernels are interleaved to match the channel-to-subword assignment of the input/output feature maps. This allows us



(a) Convolution transformed into matrix multiplication. Adapted from [12].



(b) Vectorized convolution by merging channel elements of a feature map/kernel into subwords.



(c) Our approach for vectorized convolutions. Input and output feature map elements are merged along channels, kernels are interleaved. See Sec. II-C.

Fig. 1: Comparison of different processing modes and vectorization options for convolutions.

to fully utilize computational resources with low bit width quantized NNs and simultaneously improve the data reuse during processing. Furthermore, the input and output data format and ordering are maintained throughout the whole computation of the NN. Data accesses use the native word size, which circumvents complex addressing on subword level and data alignment issues. In Sec. II-C, we illustrate in detail how convolutions are performed with our approach.

B. Multi-Bit-Width MAC Unit and Subword Permutations

Multiply-accumulate (MAC) operations dominate the NN inference phase. In Fig. 2, our proposed MAC unit design is shown. It supports computations with either full or half the original bit width, which allows us to improve the compute performance for low bit width NN computations. For simplicity, we opt for two bit widths in this example. Our method, however, is not limited to this design or configuration. Considering the two inputs a and b shown in Fig. 2, the MAC unit performs

$$acc = acc + a \cdot b \tag{1}$$

for computations using the full precision and

$$acc_{1,0} = \begin{cases} acc_1 := acc_1 + a_1 \cdot b_1 \\ acc_0 := acc_0 + a_0 \cdot b_0 \end{cases} \tag{2}$$

for computations with reduced precision. Considering the latter mode, inputs a and b are interpreted in subwords, and two MAC operations are performed simultaneously instead of one. Regardless of the desired precision, operations are always performed on subword level as shown on the left in Fig. 2. Computations requiring full precision are obtained by combining partial results:

$$a \cdot b = a_1 \cdot b_1 \cdot 2^n + a_1 \cdot b_0 \cdot 2^{\frac{n}{2}} + a_0 \cdot b_1 \cdot 2^{\frac{n}{2}} + a_0 \cdot b_0 \ , \tag{3}$$

where n is the bit width and multiplications with 2^i are bit shifts. Instead of using dedicated multipliers for each word width, this approach allows to reuse functional units in different operating modes, which reduces the HW cost [18]. The final output c is obtained by shifting either the entire content of the accumulator or by shifting individual subwords separately. In Fig. 2, the bit width mode for the addition, multiplication, and shifting is set by the configuration signal *config$_{MBW}$*.

As outlined in Sec. II-A, rather than simply vectorizing computations, we consider the semantic context of processed data to improve data reuse (details follow in Sec. II-C). To this end, we add a subword permutation stage for one of the input operands (see input operand a in Fig. 2). Based on the subword permutation configuration signal *config$_{SP}$*, subwords of input a are swapped. Consequently, (2) is extended to (note the swapped a_1 and a_0)

$$acc_{1,0} = \begin{cases} acc_1 := acc_1 + a_0 \cdot b_1 \\ acc_0 := acc_0 + a_1 \cdot b_0 \end{cases} \text{if swap active} \\ \begin{cases} acc_1 := acc_1 + a_1 \cdot b_1 \\ acc_0 := acc_0 + a_0 \cdot b_0 \end{cases} \text{otherwise.} \tag{4}$$

With our design we address the following key points:

Efficiency. Accesses to off-chip memory have a few orders of magnitude higher energy costs compared to computations [19] and executing NNs typically involves more memory accesses than computations [12]. Consequently, improving data reuse contributes to increasing the energy efficiency significantly. Using subword permutations, we can use loaded data multiple times and fuse computations that would otherwise occur sequentially and involve additional data loads. This effectively reduces the number of data transfers.

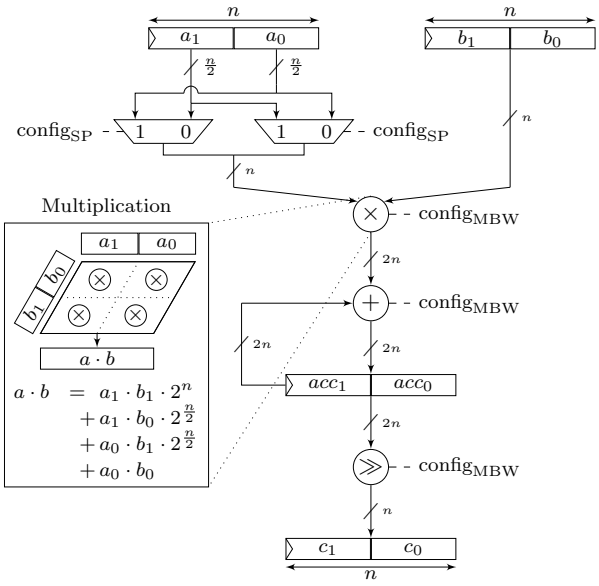


Fig. 2: Schematic of our proposed MAC unit. Control signals *config$_{SP}$* and *config$_{MBW}$* set the subword permutation and multi-bit-width configuration, respectively. Depending on the configuration, MAC is performed according to (1) and (4).

Throughput. The number of performed computations increases through multi-bit-width processing (e.g., $2\times$ for two subwords, etc.). Furthermore, we reuse already loaded data for additional computations via subword permutations. This allows us to better hide required data transfers from external memory. Consequently, idle periods of the processor during memory-bound operations are decreased, which increases the throughput.

Scalability. Increasing compute performance is usually achieved by using multiple instances of MAC units. This is only achievable up to a certain degree, until not enough bandwidth is available to keep all compute resources busy. Consequently, operations that are easily memory bound, such as depthwise separable convolutions, cannot fully benefit from performance improvements offered by increased processing capabilities. With our subword permutation method, computation efficiency can easily be increased. By reusing already loaded data for additional computations, cycles spent on processing are doubled or more (i.e., $2\times$ for two subwords, etc.), which increases the time frame in which data for subsequent operations can be loaded. This allows to better utilize the performance improvements achieved by upscaling the processor.

Control overhead. In our design, each data word contains multiple subwords and loaded data is used multiple times. Therefore, fewer data load and compute instructions have to be issued, which effectively decreases the control overhead.

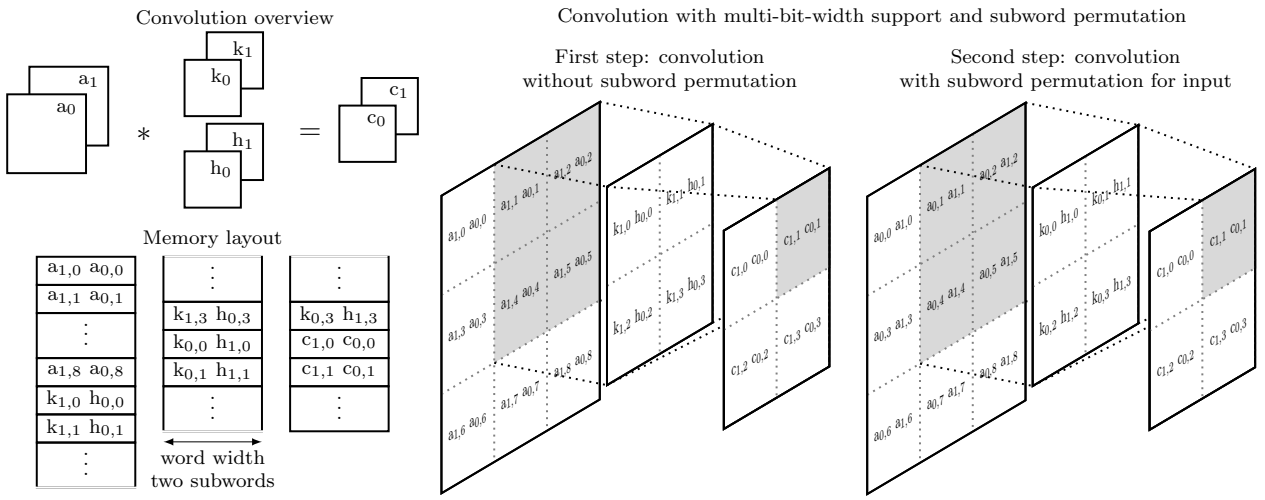


Fig. 3: Schematic showing the computational flow for multi-bit convolutions with subword permutation. Channels of input and output feature maps are merged in subwords. Channels of different kernels are interleaved in subwords to match the channel-to-subword layout of input and output feature maps. This allows us to maintain the channel-to-subword assignment throughout the whole computation of a NN. Each data word consists of two subwords $[x_{i,j}\ y_{i,j}]$, where i and j denote the channel and element indices, respectively.

While we focus on a MAC unit design of vector processors in this paper, our method is generalizable. Customizable processors such as RISC-V can also benefit from our proposed architecture. Furthermore, the design is not limited to ASICs. FPGA implementations usually utilize DSP slices for fast and efficient computations. Since the subword permutation stage does not require access to the MAC unit's data path, permutations can be performed before passing data to a DSP slice, making our design suitable for a broad range of applications. For ASIC implementations, the subword permutation stage for operand a in Fig. 2 can be integrated directly into the multiplier. Since partial products for all subword combinations are computed (see (3) and Fig. 2 left), subword permutations can simply be performed by selecting the appropriate partial product based on the configuration signal $config_{SP}$.

C. Computational Flow for Convolutions

In the following, we explain how a convolution is performed using the proposed microarchitecture. Fig. 3 illustrates the computational flow of a convolution layer where subword sizes are maintained throughout the whole computation, i.e., the subword size of the input is equal to the size of the output of the layer and the channel-to-subword assignment remains the same. For simplicity, we consider a 3×3 input a, a 2×2 output c, and 2×2 kernels k and h. Channels of the input a and output c are merged in subwords as shown in the memory layout on the bottom left. Each entry has two subwords $[x_{i,j}\ y_{i,j}]$, where y is the lower subword, x the higher subword. The channel and element indices are denoted by i and j, respectively. The channels of kernels k and h are not merged in subwords but are interleaved to match the merged channel and subword layout of a and c. We explain our method using two channels and two subwords in this work. However, for layers with more channels and/or a HW architecture that supports more than two subwords, channels are merged in a similar fashion.

The convolution is then performed in multiple steps, depending on the number of subwords. For two subwords, it is performed once without subword permutation and once with a halfword swap for input a as shown on the right in Fig. 3. Note that a is loaded only *once* from main memory and is used *twice* to perform the convolution over *two* input channels for *two* kernels. Fig. 3 shows that our proposed mapping maintains the channel-to-subword assignment throughout the convolution. Subsequent layers can immediately use the output data for further processing. Furthermore, the mapping of the NN on HW can remain unchanged. Therefore, our MAC unit design can be used as a drop-in replacement in existing architectures. From a programming perspective, each data word now simply contains two channels instead of one.

In contrast to SIMD processing, our computations are not simply vectorized. We leverage the semantic context of processed data and consider the channel-to-subword assignment during processing. Consequently, we can effectively reuse data, fuse multiple computations, and reduce redundant loading of data. Moreover, with subword permutations we maintain the channel-to-subword assignments during NN processing. This eliminates data alignment issues between individual layers. Our method requires only preprocessing of the first layer's input data such that channels are merged into subwords. Subsequent layers automatically have the correct data format and ordering. Merging and interleaving of kernels is done offline and therefore they do not require complex data accessing schemes during runtime.

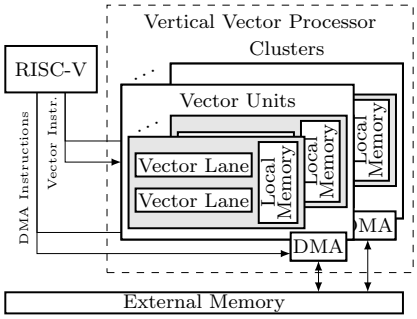


Fig. 4: Schematic of the V^{2}PRO accelerator system [5], [7].

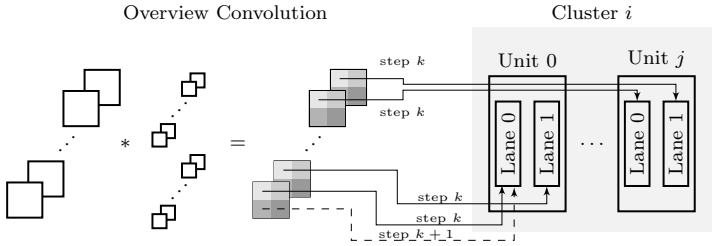


Fig. 5: Mapping of convolutions on our targeted HW [5], [7]. Arrows indicate assignment of feature map segments to vector lanes. As there are more usually more segments to process than there are lanes, each lane is responsible for multiple segments. For clarity, segment for step $k+1$ is only shown for one lane.

III. Experiments

A. Experimental Setup

To validate our proposed design, we conduct a design space exploration using our scalable vector processor accelerator V^{2}PRO [5], [7] as a hardware target. In Fig. 4 an overview of the accelerator system is shown. The vector processor is made up of clusters, which consist of vector units. Each unit has two vector lanes, the actual processing elements, where our proposed design is integrated. Vector units have a configurable local memory, which can be accessed by the lanes in the unit. Additionally, each vector lane has a register file scratchpad. In our experiments we adapt the number of clusters and units. A RISC-V processor orchestrates the computation task and is responsible for generating DMA and vector processor instructions that are broadcast to all clusters, units, and lanes. Data can also be broadcast, which reduces the required data transfers. Our results are obtained using a custom cycle accurate simulator of the vector processor [5]. We model instruction and data transfer cycle counts in our simulations. For our experiments, we assume ideal accesses to external memory. DMA and vector instructions are created by the RISC-V via IO write commands which are also considered in our simulation.

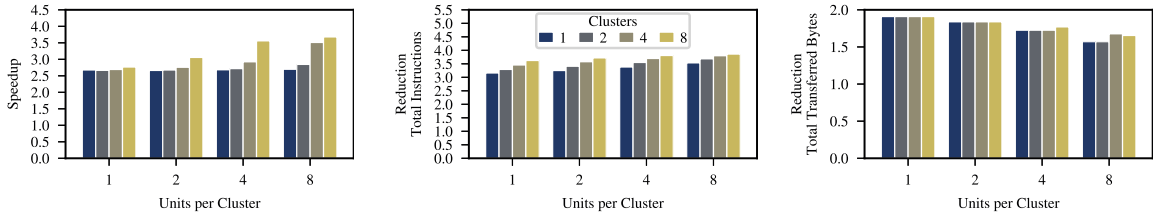
We implement MobileNet V1 [1] and VGG16 [20] in TensorFlow to evaluate our design. VGG16 uses solely standard 2D convolution layers. MobileNet has been designed for mobile applications and therefore employs depthwise separable convolutions, which involve fewer weights and MAC operations than the standard variant. It is a deeper network than VGG16 and has layers with much more filters. We trained VGG16 on CIFAR-10 [21] and achieve a classification accuracy of 91%. After training, its weights and activations are quantized from floating-point to 8-bit fixed-point without accuracy loss using the method outlined in [9]. For MobileNet, we use the publicly available model trained on ImageNet [22]. In our experiments we also consider an 8-bit quantization for weights and activations for this NN. This configuration has shown to be sufficient to maintain the network's full-precision accuracy of 70.5% [8]. We use the different architectures and layer types of both networks to assess the applicability of our method for different NNs.

Different cluster and unit configurations are evaluated in terms of performance and efficiency based on the number of clock cycles, instructions, and transferred bytes from and to the accelerator that are required for executing the NN. We compare our design with a standard MAC unit without multi-bit processing and subword permutations. For a fair comparison of data transfers, we consider a word size of 8-bit as our base line. In experiments involving our new MAC architecture, we consider V^{2}PRO's native word size of 16-bit such that two subwords with eight bit each are used during multi-bit processing. Since our proposed design focuses on optimizing convolutions, we evaluate only the convolutional part of the networks (MobileNet $\approx 568 \cdot 10^6$ MAC operations, 27 layers: $1\times$ standard 2D convolution, $13\times$ depthwise, $13\times$ pointwise convolutions. VGG16 $\approx 313 \cdot 10^6$ MAC operations, $13\times$ standard 2D convolution layers). We map convolutions on the HW using an output stationary data flow. Larger feature maps that do not fit in on-chip memory are split into multiple segments. Individual segments of a layer's output are then computed on each vector lane as shown in Fig. 5. Since there are usually more segments to process than there are vector lanes, each lane is responsible for multiple segments. We use double buffering to preload activations and weights required for subsequent computations.

B. Results

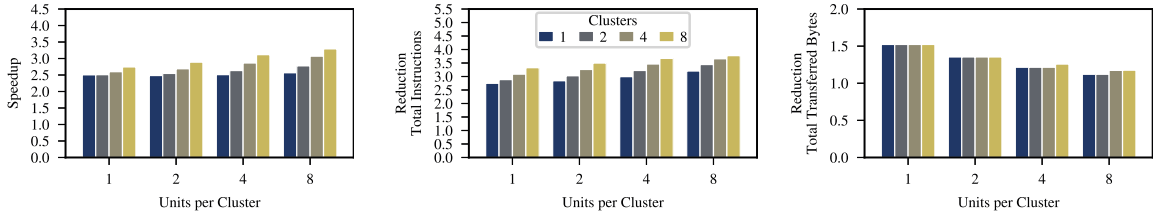
The results of our design space exploration are shown in Figs. 6 and 7. In each plot, the number of vector units is shown on the x-axis and the number of clusters is indicated by different colored bars. The y-axis indicates the improvement for our evaluated metrics speedup, control overhead, and data transfers. All results are also listed in Tab. I.

Figs. 6a and 7a evaluate the speedup for MobileNet and VGG16. For MobileNet, we achieve an average speedup of $2.9\times$ and up to $3.9\times$ compared to our baseline. Regardless of the HW configuration, a minimum speedup of $2.7\times$ is



(a) Speedup (b) Instructions (c) Data transfers

Fig. 6: Comparison of our design vs. the baseline for MobileNet. Higher is better.



(a) Speedup (b) Instructions (c) Data transfers

Fig. 7: Comparison of our design vs. the baseline for VGG16. Higher is better.

TABLE I: Improvements of our method compared to the baseline. All results in terms of $n\times$ improvement.

Cluster Count		1C				2C				4C				8C			
Unit Count		1U	2U	4U	8U	1U	2U	4U	8U	1U	2U	4U	8U	1U	2U	4U	8U
MobileNet	Speedup	2.7	2.7	2.7	2.7	2.7	2.7	2.7	2.8	2.7	2.8	2.9	3.5	2.8	3.1	3.6	3.7
	Instructions	3.2	3.3	3.4	3.5	3.3	3.4	3.5	3.7	3.5	3.6	3.7	3.8	3.6	3.7	3.8	3.9
	Data Transfers	1.9	1.8	1.7	1.6	1.9	1.8	1.7	1.6	1.9	1.8	1.7	1.7	1.9	1.8	1.8	1.7
VGG16	Speedup	2.5	2.5	2.5	2.6	2.5	2.5	2.6	2.8	2.6	2.7	2.9	3.1	2.7	2.9	3.1	3.3
	Instructions	2.7	2.8	3.0	3.2	2.9	3.0	3.2	3.4	3.1	3.3	3.5	3.7	3.3	3.5	3.7	3.8
	Data Transfers	1.5	1.4	1.2	1.1	1.5	1.4	1.2	1.1	1.5	1.4	1.2	1.2	1.5	1.4	1.3	1.2

achieved for MobileNet. For VGG16, our method enables an average speedup of $2.7\times$ with a maximum of $3.3\times$ and a minimum of $2.5\times$. Increasing the cluster and unit counts increases the compute performance, but also requires more data transfers to keep all lanes busy. The extended computation time that is obtained using subword permutations helps hiding data transfers and thus improving performance. Consequently, we are able to better capitalize on performance improvements that additional HW resources offer. Without this feature, the speedup would be limited to $2\times$ as we are processing two subwords simultaneously.

The reduction of the total issued DMA and vector instructions is shown in Figs. 6b and 7b. For MobileNet, the instruction count is reduced at least $3.2\times$ and up to $3.9\times$ for larger HW configurations (on average $3.6\times$). VGG16 sees a similar improvement with a reduction of at least $2.7\times$ and up to $3.8\times$ ($3.3\times$ on average). In general, the total instruction count decreases for a larger number of clusters and units since instructions can be broadcast. Our method allows us to merge convolution operations of two kernels over two input channels that would otherwise be performed sequentially and involve additional instructions. We can omit those and therefore decrease the control overhead further.

Figs. 6c and 7c show the reduction of data transfers. We can reduce the transferred bytes up to $1.9\times$ for MobileNet, with the minimum and average being $1.6\times$ and $1.8\times$, respectively. For VGG16, the transfers are reduced up to $1.5\times$, with a minimum of $1.1\times$ and an average of $1.3\times$. Overall, improvements are more prominent for smaller HW configurations and decrease for larger ones. Considering a sufficiently large HW configuration such that all output channels of each layer of the NN can be processed on a separate lane (cf. mapping strategy illustrated in Fig. 5). In this case, input data would have to be loaded exactly once such that the total transferred bytes would be the same for both our method and the baseline. Consequently, the resulting reduction would be equal to $1.0\times$, i.e., no reduction. This effect can be seen for VGG16. Layers of VGG16 have fewer channels than MobileNet, resulting in an average reduction around $1.3\times$. For comparison, layers of VGG16 start with 64 channels and have 512 in deeper layers, whereas layers of MobileNet start with 32 channels and reach 1024 for deeper ones. For layers with more channels than available lanes, input data has to be loaded multiple times. Since we perform the convolution over two input channels for two kernels simultaneously in subwords, we can decrease redundant loading of input data. For MobileNet, the average reduction is with $1.8\times$ close to the optimal achievable reduction of $2.0\times$.

C. Discussion

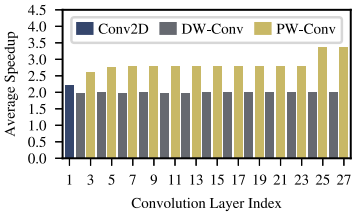
To investigate the benefit of subword permutations, we evaluate the layer-wise performance improvements for each NN separately. Figs. 8a and 8b show the average speedup for all evaluated HW configurations.

For depthwise layers of MobileNet (Fig. 8a), a speedup of $2\times$ is achieved for all configurations. Since kernels in depthwise (DW) layers operate only on a single channel, subword permutations cannot be used. The achieved acceleration is in line with what one would expect since we are using two subwords in our simulations and thus two operations are performed simultaneously instead of one. Pointwise (PW) layers operate on multiple input channels and can benefit from the subword permutation technique. Computations are at least $2.6\times$ faster using our design. The improved data reuse and the associated fused layer computations can accelerate computations beyond what the number of subwords would suggest. Pointwise layers perform only a scalar multiplication for each channel, therefore fewer cycles are required for completing the computation compared to standard 2D convolutions. As a result, it is difficult to hide data transfers which is why this layer type is easily memory bound instead of compute bound. This is especially noticeable for deeper layers with small feature map sizes which results in poorly utilized HW. Subword permutations can extend the cycles spent on processing and thus increase the time frame in which required data transfers for subsequent operations can be hidden. This effect can be seen in Fig. 8a, where deeper layers enjoy an increasing speedup. On average, the speedup for pointwise layers is $2.9\times$ and reaches up to $3.4\times$ for the last two layers.

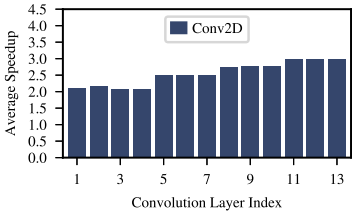
Fig. 8b shows the layer-wise results for VGG16. On average, the speedup is $2.6\times$. The 3×3 kernels of VGG16 require $9\times$ more MAC operations than the pointwise layers of MobileNet. Additionally, the first few layers have large feature maps and are therefore not as easily memory bound. Consequently, the computation is only $2.1\times$ faster. For deeper layers, the speedup increases due to the decreased feature map sizes. On average, convolution layers of VGG16 have a speedup of $2.6\times$ and up to $3.0\times$ in the last layers.

The results confirm that our method is advantageous for different types of convolutional layers. Overall, the benefits are more pronounced for MobileNet but also standard 2D convolutions enjoy a significant acceleration. Consequently, our design can be applied for all current CNN architectures.

Overall, the results show that our proposed design enables a significant speedup and reduces required data transfers and control overhead. Furthermore, we can better take advantage of performance improvements offered by increased computational resources. To the best of our knowledge, we are the first to consider and exploit the semantic relationship of processed data in our data flow using subword permutations during NN computations. This allows us to significantly enhance utilization of compute resources without altering the mapping of the NN on HW. Considering hardware cost, subword permutations are only required for one operand and can be realized using MUXes which keeps the HW overhead low. The main added complexity compared to a conventional MAC unit design lies in the multi-bit-width support. By reusing multiple low-precision functional units to compose results requiring larger bit widths, the HW overhead is kept minimal [18]. Furthermore, since our method has shown to achieve performance improvements of up to $3.7\times$, the added HW resources can be considered a reasonable investment. In future work, we will investigate the trade-off between increased HW complexity and the obtained performance and efficiency gains.



(a) Speedup for MobileNet.



(b) Speedup for VGG16

Fig. 8: Layer-wise average speedup for our design vs. the baseline. Higher is better.

While our experiments focus on a vector processor architecture, also other NN accelerators can benefit from our proposed multi-bit-width MAC unit design and subword permutations. For array-like accelerators [14], data is usually exchanged between processing elements (PEs) in the array each clock cycle. When integrating our design, multiple cycles are spent operating on the same data using subword permutations before passing it to neighboring PEs. The general mapping strategy for computing the NN remains unchanged.

IV. Related Work

The authors in [23] propose a signal processor for speech enhancement. Their flexible MAC unit is capable of accelerating FFTs and convolutional neural networks. Similar to our work, they support a native word size of 16-bit. However, they consider 16-bit activation and weights for NNs. Consequently, current low bit-width NNs do not fully utilize the HW resources and cannot benefit from improved acceleration.

In [10], [11] flexible MAC unit architectures are proposed. The authors focus on mixed-precision NNs, where layers of a NN are quantized differently. The authors in [10] compare their architecture Bit Fusion to Eyeriss [14], a NN accelerator with a native word size of 16-bit and achieve an average speedup of $3.9\times$. Compared to our approach, their speedup stems from mixed-precision computations with 2, 4, 8, or 16 bits, which increases the number of computations performed

in parallel and thus the throughput. Bit Fusion is further optimized in [11], where an additional 50% performance increase is achieved. Our results show that we achieve a similar acceleration for our architecture without resorting to mixed-precision computations. Furthermore, in [10], [11] computations are basically vectorized. Improving data reuse and support for depthwise separable convolutions is not considered, which affects the performance for HW-friendly NN architectures.

In general, however, the designs in [10], [11] and ours are not mutually exclusive. In fact, they can complement each other to extend processing flexibility and achieve further improvements in performance and efficiency. Our architecture can also be extended to support additional bit widths as well as mixed-precision NNs to take advantage of the improvements achieved in [10], [11]. When subsequent layers in mixed-precision NNs use the same quantization configuration, our method allows the channel-to-subword mapping to be maintained, which eliminates data alignment issues and enables better data reuse. For example, adding support for 4-bit computations in our 16-bit architecture allows four operations to be executed simultaneously, where input data can be reused three times. Jointly optimizing the HW architecture, quantization levels, and mapping of NN computations on HW (e.g., when switching to a different precision is beneficial), is a promising direction for future work.

V. Conclusion

In this paper, we optimize a MAC unit architecture to enhance support for low bit width NN computations by considering the semantic relationship of data during processing. Our design space exploration on a vector-based processor with two NNs shows that our proposed design can optimize multiple metrics simultaneously. Furthermore, the achieved benefits are not limited to normal 2D convolutions. Especially mobile-friendly depthwise separable convolution can be accelerated significantly, which is often not addressed in related approaches. In future work, we will evaluate hardware related metrics on an FPGA and investigate the trade-off between increased HW complexity and the obtained improvements presented in this paper.

Acknowledgment

This work is supported by the German federal ministry of education and research (BMBF), project ZuSE-KI-AVF (grant no. 16ME0062).

References

[1] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," in *arXiv preprint arXiv:1704.04861*, 2017.

[2] Y.-H. Chen, T.-J. Yang, J. S. Emer, and V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[3] T. Dietzen, E. De Sena, and T. van Waterschoot, "Low-Complexity Steered Response Power Mapping Based on Nyquist-Shannon Sampling," in *2021 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE, Oct. 2021, pp. 206–210.

[4] S. Sun, A. P. Petropulu, and H. V. Poor, "MIMO Radar for Advanced Driver-Assistance Systems and Autonomous Driving: Advantages and Challenges," *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 98–117, Jul. 2020.

[5] G. B. Thieu *et al.*, "ZuSE-KI-AVF: Application-Specific AI Processor for Intelligent Sensor Signal Processing in Autonomous Driving," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023.

[6] M. Weißbrich, A. García-Ortiz, and G. Payá-Vayá, "Comparing vertical and horizontal SIMD vector processor architectures for accelerated image feature extraction," *Journal of Systems Architecture*, vol. 100, p. 101647, Nov. 2019.

[7] S. Nolting, F. Giesemann, J. Hartig, A. Schmider, and G. Payá-Vayá, "Application-Specific Soft-Core Vector Processor for Advanced Driver Assistance Systems," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–2.

[8] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling, "Data-free Quantization Through Weight Equalization and Bias Correction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1325–1334.

[9] S. Vogel, J. Springer, A. Guntoro, and G. Ascheid, "Self-Supervised Quantization of Pre-Trained Neural Networks for Multiplierless Acceleration," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1094–1099.

[10] H. Sharma *et al.*, "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 764–775.

[11] S. Ghodrati, H. Sharma, C. Young, N. S. Kim, and H. Esmaeilzadeh, "Bit-Parallel Vector Composability for Neural Acceleration," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[12] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[13] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. ACM, Jun. 2017, pp. 1–12.

[14] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

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

[16] A. Vasudevan, A. Anderson, and D. Gregg, "Parallel Multi Channel Convolution using General Matrix Multiplication," in *2017 IEEE 28th international conference on application-specific systems, architectures and processors (ASAP)*. IEEE, 2017, pp. 19–24.

[17] S. Chetlur *et al.*, "cuDNN: Efficient Primitives for Deep Learning," *arXiv preprint arXiv:1410.0759*, 2014.

[18] L. Gerlach, G. Payá-Vayá, and H. Blume, "An area efficient real- and complex-valued multiply-accumulate SIMD unit for digital signal processors," in *2015 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2015, pp. 1–6.

[19] M. Horowitz, "Computing's Energy Problem (and what we can do about it)," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 10–14.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[21] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," Tech. Rep., 2009.

[22] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[23] Y.-C. Lee, T.-S. Chi, and C.-H. Yang, "A 2.17-mW Acoustic DSP Processor With CNN-FFT Accelerators for Intelligent Hearing Assistive Devices," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 8, pp. 2247–2258, Aug. 2020.