2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-7652-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/HPCA56546.2023.10071076

Mix-GEMM: An efficient HW-SW Architecture for Mixed-Precision Quantized Deep Neural Networks Inference on Edge Devices

Enrico Reggiani [*†], Alessandro Pappalardo [‡], Max Doblas [*†],
Miquel Moreto [*†], Mauro Olivieri [†§], Osman Sabri Unsal [†], Adrián Cristal [*†]

[*] Polytechnic University of Catalonia
[†]Barcelona Supercomputing Center
[‡]AMD AECG Research Labs, Dublin, Ireland
[§]Sapienza University of Rome, Roma, Italy
[†]{*enrico.reggiani, max.doblas, miquel.moreto, osman.unsal, adrian.cristal*}*@bsc.es*
[‡]{*alessand*}*@amd.com*
[§]{*mauro.olivieri*}*@uniroma1.it*

***Abstract*— Deep Neural Network (DNN) inference based on quantized narrow-precision integer data represents a promising research direction toward efficient deep learning computations on edge and mobile devices. On one side, recent progress of Quantization-Aware Training (QAT) frameworks aimed at improving the accuracy of extremely quantized DNNs allows achieving results close to Floating-Point 32 (FP32), and provides high flexibility concerning the data sizes selection. Unfortunately, current Central Processing Unit (CPU) architectures and Instruction Set Architectures (ISAs) targeting resource-constrained devices present limitations on the range of data sizes supported to compute DNN kernels.**

This paper presents *Mix-GEMM*, a hardware-software co-designed architecture capable of efficiently computing quantized DNN convolutional kernels based on byte and sub-byte data sizes. *Mix-GEMM* accelerates General Matrix Multiplication (GEMM), representing the core kernel of DNNs, supporting all data size combinations from 8- to 2-bit, including mixed-precision computations, and featuring performance that scale with the decreasing of the computational data sizes. Our experimental evaluation, performed on representative quantized Convolutional Neural Networks (CNNs), shows that a RISC-V based edge System-on-Chip (SoC) integrating *Mix-GEMM* achieves up to 1.3 TOPS/W in energy efficiency, and up to 13.6 GOPS in throughput, gaining from 5.3× to 15.1× in performance over the OpenBLAS GEMM frameworks running on a commercial RISC-V based edge processor. By performing synthesis and Place and Route (PnR) of the enhanced SoC in Global Foundries 22nm FDX technology, we show that *Mix-GEMM* only accounts for 1% of the overall area consumption.

I. Introduction

Deep Neural Networks (DNNs) are currently the preferred choice for artificial intelligence and computer vision tasks in both research and industrial applications. DNNs are composed of a stack of layers, whose execution time is typically dominated by the computation of large linear algebra operations like General Matrix Multiplications (GEMMs). Optimizing DNNs represents a major challenge in many fields, in particular when targeting the deployment to hardware architectures designed for edge and mobile segments, requiring high performance but presenting tight constraints in terms of area, memory, and energy consumption.

A widespread solution aimed at decreasing this burden is *quantization*, a family of techniques designed to reduce the numerical precision required to represent the parameters of a DNN and the data computed by its layers. In particular, integer quantization focuses on deploying trained DNNs with narrow-integer formats, typically ranging from 8- down to 2-bit [35], [43], [75], rather than with the standard Floating-Point 32 (FP32) data size. Quantizing DNNs exposes a large design space, as each layer can be quantized to its own precision. Moreover, input data and parameters can also be quantized differently within a layer, resulting in mixed-precision operations. Exploring this design space allows to trade-off computational requirements against quality of results, which is a key enabler of deployment in resource-constrained devices. Indeed, reducing the precision of highlighted parameters and data decreases the memory and the bandwidth required to store and load them, allowing resource-constrained devices to support larger models, or to relax constraints around the sizing of their memory hierarchy and power envelope. Quantization also enables computing operations like GEMM at low precision, with a consequent improvement in terms of performance and energy efficiency. However, in practice, most of the current general-purpose Central Processing Unit (CPU) architectures lack adequate support for efficiently handling narrow-precision formats, as most of the Instruction Set Architectures (ISAs) neither support data sizes smaller than 8-bit, nor support mixed-precision computations. Although modern Single Instruction Multiple Data (SIMD) extensions [60] and hardware accelerators [27], [52] are increasing their support for narrow-precision data sizes and mixed-precision computations on CPU architectures, they only consider a small subset of data sizes granularities. As a result, exploiting fine-grained quantization of DNNs on modern processors does not always provide a real benefit to the actual computation performance, as quantized data have to be either saved in memory in a sub-optimal format (*i.e.*, with data sizes supported by the processor ISA), or decompressed before the actual computation exploiting costly bit-manipulation operations. Therefore, investigating hardware and software architectures

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on February 20,2025 at 08:11:03 UTC from IEEE Xplore. Restrictions apply.

capable of leveraging quantization not only to save memory space, but also to efficiently compute quantized data in terms of performance and energy efficiency, while respecting the tight area and power caps of resource-constrained devices, represents a critical research challenge for the computer architecture field.

In this paper, we propose *Mix-GEMM*, a hardware-software architecture capable of efficiently performing GEMM-based computations targeting narrow-integers. The hardware microarchitecture of *Mix-GEMM*, hosted in the processor execution stage, is built upon the *binary segmentation* technique, which allows computing high-performance SIMD operations on narrow-integer, reusing the processor Functional Units (FUs) with a negligible impact on area overhead. The key novelty of *Mix-GEMM* lies in supporting computations based on all the data size combinations between 8- and 2-bit, including mixed-precision, while exploiting high performance, that scales with the decrease of the data sizes involved in the computation. This feature allows tuning the performance of quantized DNNs inference on edge devices with high flexibility, accounting for latency, energy consumption, model accuracy, and memory footprint.

The main contributions of this paper are listed as follows:

- We design an area and energy efficient hardware accelerator, integrated into an edge processor pipeline and capable of computing mixed-precision GEMM kernels based on narrow-integers. The proposed architecture, called *μ-engine*, leverages the *binary segmentation* technique to perform from 3 to 7 MAC/cycle while reusing the processor multiplier;
- We extend the RISC-V ISA with custom instructions used to design a high-performance GEMM software library handling the *μ-engine*, allowing a fine-grained selection of the data sizes and a balance of the overall DNNs performance in terms of throughput, energy efficiency, and memory footprint;
- We integrate our *μ-engine* into a RISC-V based edge System-on-Chip (SoC), and we benchmark the performance of *Mix-GEMM* on six Convolutional Neural Networks (CNNs), namely AlexNet, VGG-16, ResNet-18, MobileNet-V1, RegNet-x-400mf, and EfficientNet-B0. For these networks, *Mix-GEMM* reaches performance ranging from 4.8 GOPS to 13.6 GOPS, and from 477.5 GOPS/W to 1.3 TOPS/W energy efficiency;
- We investigate the considered quantized CNNs in terms of TOP-1 accuracy, exploring an exhaustive set of data size combinations exploiting Quantization-Aware Training (QAT). Our evaluation shows that narrow and mixed-precision quantized CNNs can be Pareto optimal in terms of computational requirements, and show minimal accuracy losses (*i.e.*, up to 1.5%) for data sizes larger than 4-bit;
- We implement the RISC-V SoC integrating our hardware accelerator, in the Global Foundries 22nm FDX technology node, past the Place and Route (PnR) phase, showing that the proposed *μ-engine* only accounts for 1% of the total SoC area, and for an overall 2.3% on its total power consumption;

The remainder of the paper is laid out as follows. Section II introduces DNNs and the main techniques at the base of this work. Section III details *Mix-GEMM*. Section IV presents the experimental evaluation. Section V compares the main features and performance of *Mix-GEMM* with the most relevant related work. Finally, Section VI discusses the conclusions.

II. Background

A. Deep Neural Networks

In the past few years, DNNs have encountered ample success in a multitude of fields, including but not limited to computer vision, speech, and language [40] [21] [51]. We broadly define a DNN as a computational graph, where each node represents a layer.

The various types of functions computed by each layer can be typically organized in two classes: linear layers, such as *convolution* or *fully-connected*, and non-linear operations, such as *relu*, *sigmoid*, or *tanh*, with DNNs typically alternating between the two from one layer to the next one. The computation of linear layers typically represents the majority of the operations performed in a DNN. Computing DNNs normally involves two phases, *training* and *inference*. During training, the values of parameters belonging to each layer are learned from data by minimizing a loss function. During inference, the parameter values are set, and the DNN is adopted to execute the task it was designed to perform.

CNNs are a class of DNNs dominated by the computation of convolutions. Convolutions can be accelerated in a variety of ways depending on the layer dimensions [2], such as the size of the convolution kernel or the stride. While the direct approach implements it as a series of nested loops, fast algorithms like FFTs [47] or Winograd [38] exploit a numerical transformation of the input and the weights to reduce the overall number of operations. On the other hand, GEMM-based algorithms, such as the *im2row* or the *im2col* approaches [15], maps a convolution to a highly-optimized GEMM implementation.

Direct approaches typically require tuning each kernel with respect to the layer dimensions, either by providing optimized kernels for common choices of dimensions, as in libraries like cuDNN [18], or by generating code just-in-time, as in libraries such as MKL-DNN [28]. Fast algorithms are efficient only for certain dimensions of the layer, and have additional limitations when applied to quantized values [49]. On the other hand, GEMM-based approaches retain better generality, since they all call into the same pre-compiled backend for any dimensions of the layer, and thus they represent the focus of this work.

In the *im2col* GEMM-based approach, input activations and weights are reshaped and duplicated to fit into the GEMM input matrices, namely A and B. Each row of A is composed of the flattened input values that contribute to that pixel, potentially taken from a batch of multiple input images, while each column of B corresponds to flattened parameters computing a single output pixel. A direct implementation of *im2col* incurs a non-trivial overhead in terms of memory and bandwidth, because activations and weights are duplicated across A and B. However, as modern *im2col* approaches [22], [48], [72], [79] remove this overhead by implicitly composing A and B in memory, this work only focuses on the compute aspect of GEMM.

A widespread method to reduce DNNs complexity is quantization. DNNs quantization reduces the numerical precision required to represent activations and weights values. We give a brief overview of the topic in terms of what is relevant to the work presented. A more in-depth review can be found in [29] and [44].

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on February 20,2025 at 08:11:03 UTC from IEEE Xplore. Restrictions apply.

The acceleration strategy presented in this work applies to uniform affine integer quantization at inference time, which is defined as:

$$y = q(x) = clamp\left(round\left(\frac{x}{s} + z\right), y_{\min}, y_{\max}\right) \quad (1)$$

where x is the tensor to quantize, s is the *scale*, z is the *zero-point*, while $y_{\min}$ and $y_{\max}$ are defined as:

$$[y_{\min}, y_{\max}] = \begin{cases} [0, 2^{n_b} - 1] & \text{if unsigned} \\ [-2^{n_b-1}, 2^{n_b-1} - 1] & \text{if signed} \end{cases} \quad (2)$$

where n_b is the bit width to quantize to. Depending on how s, z, and b are defined, different variants emerge. The case where $z = 0$ is referred to as *symmetric* quantization, while $z \neq 0$ is *asymmetric* quantization. Quantization is named *channel-wise* if s is a 1-dimensional tensor, while it is *layer-wise* or *tensor-wise* in case s is a scalar value.

Quantization is typically adopted in a DNN through either Post-Training Quantization (PTQ) or QAT. PTQ starts from a pre-trained model in floating-point, and relies on a small amount of calibration to determine appropriate values for *scales* and *zero-points*. QAT instead models quantization at training time, allowing to compensate for quantization errors during training. While PTQ requires limited extra computation and data, and is effective at higher precisions like 7- and 8-bit, QAT carries the cost of full training, but can scale down to narrower data sizes. For this reason, QAT represents an excellent candidate to optimize DNN computations on resource-constrained devices, and it is the technique adopted in this work (see Section IV-A for more details).

B. Binary Segmentation

Binary segmentation [53] is a mathematical technique that allows reducing the arithmetic complexity of Basic Linear Algebra Subprogram (BLAS) computations based on narrow-integers data [9], [14], [62]. This technique abstracts the computation of BLAS kernels based on narrow-integers as simpler arithmetic operations, by properly representing sets of narrow-integer elements as single wider data, called *input-clusters*. Specifically, this technique allows performing SIMD computations of kernels featuring narrow-integers, exploiting the unmodified processor FUs, such as scalar multipliers and adders. As a result, *binary segmentation* is an excellent candidate to optimize the inner-product of narrow-vectors, representing the core computation of the GEMM kernel.

Binary segmentation computes the inner-product of two vectors a and b (henceforward called *μ-vectors*), having bitwidths bw_a and bw_b, as a set of multiplications among *input-clusters*. Specifically, a and b are cleverly packed to compose the *input-clusters*, whose multiplication results in the *μ-vectors* inner-product. This packing strategy follows the rule defined in Equation (3), which specifies the bitwidth of the elements packed into each *input-cluster*, called *clustering-width (cw)*. Defining the number of elements in an *input-cluster* as $input\text{-}cluster_{size}$, we determine the cw as:

$$cw \geq 1 + bw_a + bw_b + \lceil \log_2(input\text{-}cluster_{size} + 1) \rceil \quad (3)$$

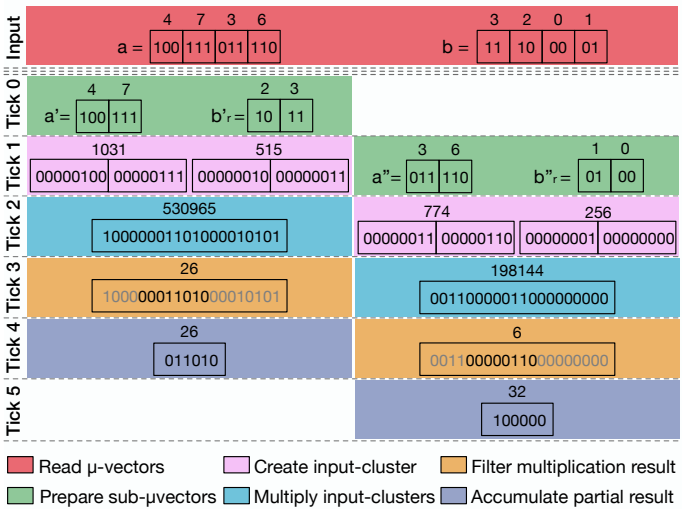

Fig. 1. Example of inner-product computation (*i.e.*, $4\times3+7\times2+3\times0+6\times1=32$) evaluated via *binary segmentation* through a pipelined approach. Each color represents a step required by *binary segmentation* to compute the inner-product. Each tick depicts the pipeline status over time.

The $input\text{-}cluster_{size}$ value is thus constrained by both the cw and the multiplier bitwidth $\text{mul}_{\text{width}}$:

$$input\text{-}cluster_{size} = \left\lfloor \frac{mul_{width}}{cw} \right\rfloor \quad (4)$$

Therefore, the $\text{mul}_{\text{width}}$ and the *μ-vectors* bitwidths determine the $input\text{-}cluster_{size}$, which translates into the arithmetic complexity reduction achievable through *binary segmentation*.

Figure 1 details the steps required by *binary segmentation* to compute the inner-product of two *μ-vectors a = [4, 7, 3, 6]* and *b = [3, 2, 0, 1]* composed of *n = 4* elements and having bitwidths bw_a and bw_b equal to 3- and 2-bit, respectively. Supposing that the example in Figure 1 exploits a multiplier having mul_{width} equal to 16-bit, Equation (3) and Equation (4) allow evaluating a cw equal to 8-bit, and a $input\text{-}cluster_{size}$ of 2 elements per *input-cluster*. As the number of elements of each *μ-vector* (*i.e.*, n) is twice as $input\text{-}cluster_{size}$, the complete inner-product computation requires applying *binary segmentation* on two separate a and b slices. To ensure continuity with the *Mix-GEMM* hardware architecture depicted in Figure 5, the example in Figure 1 express the inner-product of a and b via *binary segmentation* as a computational pipeline, whose stages follow the same color scheme of Figure 5.

In the first computational step of Figure 1 (highlighted in green), a and b are partitioned into *sub-μvectors*, having a number of elements equal to the $input\text{-}cluster_{size}$, and the elements order of each b *sub-μvector* (*i.e.*, b_r', and b_r") is reverted, according to *binary segmentation* first principles [54]. A second step (pink) converts each *sub-μvector* element to a cw-bit element, and packs it in the respective *input-cluster*. As Figure 1 shows, each *input-cluster* can be seen as a single wide integer (*i.e.*, 1031, 515, 774, and 256) having bitwidth equal to the mul_{width} (*i.e.*, 16 bit). The third step (blue) performs the *input-clusters* multiplication. A fourth step (orange) filters

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on February 20,2025 at 08:11:03 UTC from IEEE Xplore. Restrictions apply.

a slice of the multiplication output, holding the *input-clusters* inner-product, according to the following expression:

$$input\text{-}cluster_{ip} = Mul_{out}[slice_{msb} \; ; \; slice_{lsb}] \quad (5)$$

where:

$$slice_{lsb} = (input\text{-}cluster_{size} - 1) \times cw \quad (6)$$

$$slice_{msb} = slice_{lsb} + cw - 1 \quad (7)$$

Figure 1 applies Equation (5) by extracting the partial inner-products (*i.e.*, 26 and 6). Finally (grey), the partial results are accumulated to obtain the final inner-product (*i.e.*, 32). As Figure 1 shows, an inner-product of 4 elements can be performed via *binary segmentation* with only 2 16-bit multiplications and a single addition, with a consequent 2.33× arithmetic complexity reduction. Applying *binary segmentation* to a 64-bit architecture implies a computational arithmetic decrease of 5× and 13× for 8- and 2-bit data sizes, allowing thus computing inner-products with performance ranging from 3 Multiply-Accumulate (MAC)/cycle to 7 MAC/cycle exploiting a single 64-bit multiplier.

C. Efficient Matrix-Matrix Multiplication

The GEMM software library proposed in this paper is built upon the Double-precision General Matrix Multiplication (DGEMM) kernel of BLAS-like Library Instantiation Software (BLIS) [74], a state-of-the-art framework for high-performance BLAS computations [39]. BLIS exploits different compile-time strategies to improve data reusage (*e.g.*, blocking) and to optimize data movements across the memory hierarchy during the GEMM computation, guaranteeing optimal performance by minimizing the number of cache misses. Specifically, DGEMM computes a block-based multiplication between two 64-bit dense matrices A and B, with sizes $m \times k$, and $k \times n$, respectively. The multiplication result is stored in the output matrix C, having dimensions $m \times n$. To improve cache efficiency, BLIS partitions the matrices into blocks of smaller dimensions, called *panels*, stored in contiguous memory arrays. Specifically, a *panel* of the input matrix A is composed of $mc \times kc$ elements, arranged as *μ-panels* having size $mr \times kc$. Similarly, each *panel* of B holds $nc \times kc$ elements, divided into *μ-panels* holding $nr \times kc$ elements. *Panels* and *μ-panels* are constrained such that $nc \leq n$, $mc \leq m$, $kc \leq k$, $nr \leq nc$, and $mr \leq mc$. This specific *panels* reorganization assures that their elements are accessed with unit stride during the *μ-panels* computation. Each C *μ-panel*, having dimension $nr \times mr$, is evaluated in the so-called *μ-kernel*, computing the matrix-matrix multiplication between single A and B *μ-panels*. BLIS performance are optimal if its parameters (*mc*, *nc*, *kc*, *mr*, and *nr*) are correctly set. Their optimal values can be found analytically [45], and mainly depend on the target processor characteristics, such as the number of cache levels, sizes, and associativities. According to the methodology presented in [45], the C *μ-panel* is kept in the processor Register File (RF), by assigning to *mr* and *nr* values whose product does not exceed the number of RF registers. Indeed, each *μ-kernel* execution updates a different C *μ-panel* element multiple times, and therefore its partial results must be kept in memories featuring low latencies. Similarly, the *kc* dimension is set to allow storing the whole A and B *μ-panel* in the L1 cache, as their elements are reused for different *μ-kernel* iterations. Finally, *mc* is set to ensure that the A *panel* fits in the L2 cache.

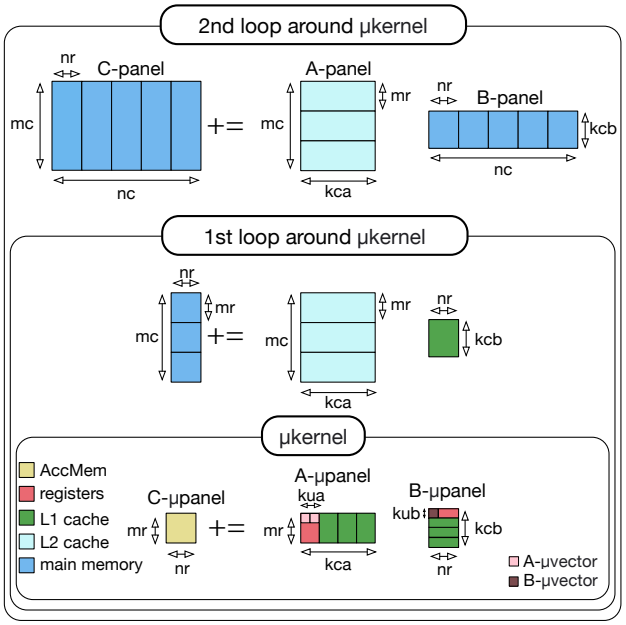

Fig. 2. Data flow and allocation of the proposed MACRO-KERNEL and μ-KERNEL procedures, built upon the BLIS implementation of DGEMM. Note that both *kca* and *kua* are twice as *kcb* and *kub*, indicating that the data size of A is two times larger than the one of B.

III. MIX-GEMM HW-SW ARCHITECTURE

This section details *Mix-GEMM*, a hardware-software architecture that allows fast GEMM computations based on narrow-integers. On the one hand, as detailed in Section III-A, the proposed software library modifies the BLIS framework to support narrow-precision integers, including all granularities of mixed-precision computations from 8- to 2-bit data sizes, supporting the complete range of bitwidths typically exploited in quantized DNNs. On the other hand, Section III-B describes the *Mix-GEMM* hardware microarchitecture, called *μ-engine*. The *μ-engine* exploits the *binary segmentation* technique to perform SIMD MAC operations through the unmodified 64-bit processor scalar multiplier, and to efficiently handle narrow mixed-precision GEMM computations, at a negligible cost in terms of area and power consumption.

A. μ-engine GEMM Software Library

We build our narrow-precision GEMM software library on top of the DGEMM algorithm implemented in the BLIS framework, described in Section II-C. The proposed library leverages BLIS to efficiently move vectors of narrow-precision data through the processor cache hierarchy. Our GEMM library keeps the input matrices A and B compressed over their common k dimension, in chunks ranging from 8 to 32 elements, for 8- and 2-bit data sizes, respectively. Each chunk of compressed elements composes a *μ-vector*, introduced in Section II-B.

As a result, the proposed software library leverages cache-friendly data movements of the BLIS-based DGEMM algorithm, abstracting each *μ-vector* as a single 64-bit element. This data organization allows handling non-standard data sizes without extending the processor ISA, increasing the efficiency of quantized DNN computations from different

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on February 20,2025 at 08:11:03 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 Mix-GEMM pseudo-algorithm

```
1: procedure μ-KERNEL(A_μp, B_μp, C)
2:     for kca/kua iterations do                          ▷ same as kcb/kub
3:         for i = 0 → nr − 1 do
4:             for j = 0 → mr − 1 do
5:                 for k = 0 → kua − 1 do
6:                     A_μvector = A_μp[k + mr ∗ j]
7:                     B_μvector = k < kub ? B_μp[i + nr ∗ k] : 0
8:                     bs.ip(A_μvector, B_μvector)
9:         LoadNextAddress(A_μp)                          ▷ next kua × mr elements
10:        LoadNextAddress(B_μp)                          ▷ next kub × nr elements
11:    for i = 0 → nr − 1 do                              ▷ Get output from AccMem
12:        for j = 0 → mr − 1 do
13:            C_μp[i, j] = bs.get(j + i ∗ mr)
14:    UpdateC(C_μp, C)
15: procedure MACRO-KERNEL(A_p, B_p, C)
16:    for nc/nr iterations do
17:        B_μp = CreateμPanel(B_p)
18:        for mc/mr iterations do
19:            A_μp = CreateμPanel(A_p)
20:            μ-KERNEL(A_μp, B_μp, C)
21: procedure M-GEMM
22:    bs.set(aX − wY)                                    ▷ Load X-bit Y-bit configuration
23:    for n/nc iterations do
24:        for ka/kca iterations do                       ▷ same as kb/kcb
25:            B_p = CreateBPanel()
26:            for m/mc iterations do
27:                A_p = createAPanel()
28:                MACRO-KERNEL(A_p, B_p, C)
```

perspectives. First, it enables keeping the DNN activations and weights compressed in main memory (even if their data sizes are not supported by the processor ISA), thus allowing to deploy bigger DNNs on resource-constrained devices. Second, it significantly reduces the number of memory instructions required to perform the GEMM computation, directly impacting performance and energy consumption.

Figure 2 details the dimensions and the memory locations of the matrices used in the *μ-engine* GEMM software library. In Figure 2, we follow the approach proposed in [45] and detailed in Section II-C to partition *panels* and *μ-panels* into a specific level of the processor memory hierarchy. Blocks of elements featuring fewer data reuse are kept in main memory or in the L2 cache, while the ones reused more often are sized to fit either the L1 cache or the processor RF. To further improve data locality, *Mix-GEMM* defines a further level in the memory hierarchy, called Accumulator Memory (AccMem) and held inside the *μ-engine*. The AccMem locally stores an entire *C μ-panel* having dimension $mr \times nr$ elements, allowing to further increase data locality, and to free the RF registers formerly reserved to the *C μ-panel*, which are instead allocated to slices of the *A* and *B μ-vectors*, avoiding thus to load the same data from cache multiple times.

Algorithm 1 shows the pseudo-code of the proposed BLIS-based library implementation, whose top-function is represented by the M-GEMM procedure. The M-GEMM procedure loads the current *μ-engine* configuration through a custom RISC-V instruction called `bs.set()` (line 22), and then splits the *A* and *B* input matrices in *panels*, holding $mc \times kca$ and $nc \times kcb$ *μ-vectors*, respectively (line 25 and 27). Note that we introduce two separate *k*-dimensions for the *A* and *B panels*, (*i.e.*, *kca* and *kcb*), to account for mixed-precision computations, where the input matrices show different *k* values. Specifically, in Figure 2,

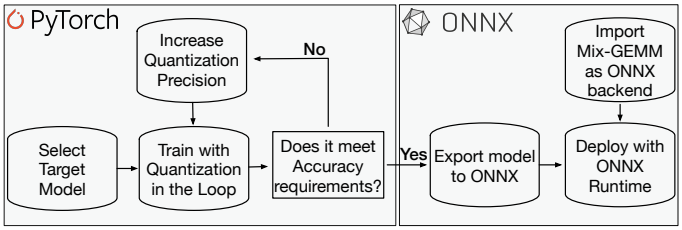

Fig. 3. *Mix-GEMM* training and inference workflow diagram.

kca is two times *kcb*, implying that the compressed elements stored in *A* have twice the data size as the compressed elements of *B*. While the M-GEMM procedure is used to partition the *A* and *B* matrices in *panels*, the MACRO-KERNEL procedure of Algorithm 1 splits each *panel* into *μ-panels* (lines 17 and 19), which are then forwarded to the μ-KERNEL procedure (line 20). The μ-KERNEL computes the actual matrix-matrix multiplication between an *A μ-panel*, composed of $mr \times kca$ *μ-vectors*, and a *B μ-panel*, composed of $kcb \times nr$ *μ-vectors*, thus creating a *C μ-panel* of $mr \times nr$ elements. Each innermost iteration of the μ-KERNEL loads a *μ-vector* pair from the *μ-panels* (lines 6 and 7), and forwards it to the *μ-engine* through a `bs.ip()` instruction (line 8), that computes its inner-product. Each inner-product is stored in the AccMem and, once the entire *μ-panels* have been issued to the *μ-engine* through `bs.ip()` instructions (lines 2 to 10), the result is collected from the AccMem using $mr \times nr$ `bs.get()` instructions (lines 11 to 13), and accumulated in the output matrix *C* (line 14). The `bs.set()`, `bs.ip()`, and `bs.get()` instructions are implemented as single-cycle instructions, and exploited in the *μ-engine* GEMM library as intrinsics extending the RISC-V ISA.

Note that in case of mixed-precision computations, each *μ-vector* pair holds a different number of narrow-elements. For example, for a mixed-precision configuration where the *A* and *B* input matrices are composed of 8- and 2-bit data (*i.e.*, 8 and 32 elements per *μ-vector*, respectively), a single *B μ-vector* requires four *A μ-vectors* to issue the same number of narrow-elements to the *μ-engine*. As a result, to balance the number of elements effectively computed by each inner-product, the number of *A* and *B μ-vectors* sequentially issued to the *μ-engine* could differ for some data size configurations. Therefore, we extend BLIS with two parameters, namely *kua* and *kub*, aimed at selecting the actual number of subsequent *μ-vectors* on each innermost *μ-kernel* iteration.

Examples of mixed-precision computations requiring different combinations of *kua* and *kub* are reported in Figure 4. Each example considers a different combination of *A* and *B* data sizes (*e.g.*, the data sizes of DNN activations *aX* and weights *wY*). As the *a8-w8* configuration is composed of *μ-vectors* holding 8-bit for both input matrices, *kua* and *kub* are equal (*e.g.*, set to 4). As a result, each μ-KERNEL innermost execution (lines 5 to 8 in Algorithm 1) issues 4 *μ-vector* pairs (*i.e.*, 32 narrow-elements) to the *Mix-GEMM μ-engine*, which computes their inner-product. On the other hand, in both the *a8-w6* and *a6-w4* configurations of Figure 4, the number of narrow-elements held in a single *A μ-vector* is not equal to the one stored in a single *B μ-vector*. Consequently, *kua*

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on February 20,2025 at 08:11:03 UTC from IEEE Xplore. Restrictions apply.

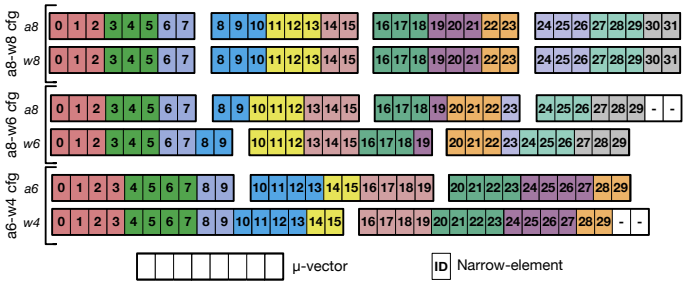

Fig. 4. Representation of three activation-weight configurations. Each *μ-vector* holds a different number of elements, depending on the element data size. Different colors represent different *μ-engine* execution cycles (*i.e.*, different selected *sub-μvectors*).

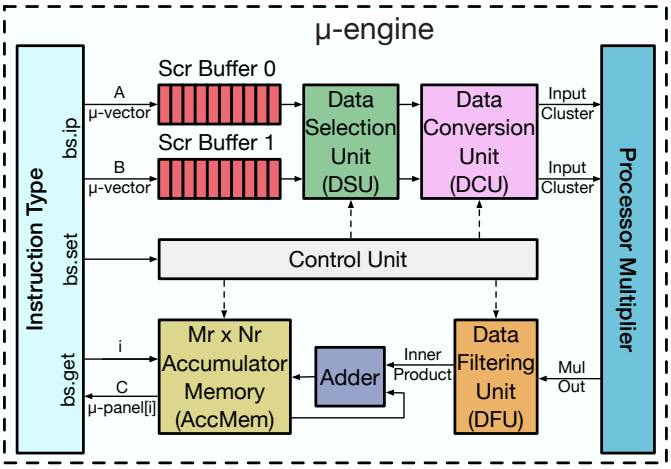

Fig. 5. *μ-engine* architecture, integrated in the processor FU. Different colors represent compute or memory unit details, according to Figure 1 (green, pink, blue, orange, and grey), or Figure 2 (red and yellow).

and *kub* are set to guarantee a match in the overall number of narrow-elements issued to the *μ-engine*. Specifically, the *a8-w6* configuration features *kua* and *kub* equal to 4 and 3, while the *a6-w4* example sets *kua* and *kub* equal to 3 and 2.

The proposed software library can be easily integrated as an additional backend in the ONNX framework [7], and used to accelerate DNN inference through the ONNX Runtime engine. Figure 3 reports the workflow diagram of *Mix-GEMM*. The target Pytorch model is trained exploiting QAT, gradually increasing activations and weights data sizes (either *per-layer* or *per-network* depending on the selected granularity) until the target accuracy is met. The quantized model is then converted to an ONNX model, and deployed through ONNX Runtime exploiting *Mix-GEMM* as a backend to accelerate the model layers based on BLAS computations.

B. μ-engine Hardware Architecture

The *μ-engine* architecture, depicted in Figure 5, is composed of a computational pipeline, whose stages perform a specific *binary segmentation* step. The *μ-engine* is fully integrated in the scalar processor execution stage as an additional FU, and its functionalities are completely integrated with the processor pipeline. Note that, to graphically describe the functionality of each *μ-engine* component, we adopt the same color scheme for Figure 5 and the *binary segmentation* example of Figure 1.

As discussed in Section III-A, we extend the RISC-V ISA with three R-type instructions, named `bs.set()`, `bs.ip()`, and `bs.get()`. The `bs.set()` instruction is issued to the *μ-engine* once for the entire GEMM computation, and it is used to configure its *Control Unit*. The parameters used to configure the *Control Unit* either provide details about the incoming *μ-vectors*, such as their data sizes and computation type (*i.e.*, signed or unsigned), or specify *binary segmentation* related constraints, such as the *input-cluster*$_{size}$, the *cw*, the inner-product length, and the slice of data to extract from the multiplication output.

The *Control Unit* requires a single clock-cycle to be reconfigured, and thus introduces a negligible overhead in the computation with respect to the complete GEMM execution. As a result, the data sizes of weights and activations can be easily tuned for each layer of the model, providing a further degree of freedom when exploring the data size configurations, and allowing selecting the best trade-off between performance and accuracy.

Once the *Control Unit* is properly configured, the *μ-engine* GEMM library starts issuing multiple `bs.ip()` instructions to perform the computation. The source operands of each `bs.ip()` instruction (*i.e.*, a *μ-vector* pair) are buffered in two separate *Source Buffers*, and then handled by the Data Selection Unit (DSU), whose main purpose is to select the appropriate number of narrow-elements (*i.e.*, the *sub-μvectors*) on every clock cycle. Figure 4 reports three examples of DSU activity. For each configuration, different colors represent different execution cycles. On each cycle, *sub-μvectors* starting from the element with ID 0 are selected by the DSU. The maximum number of elements selected per cycle is equal to the *input-cluster*$_{size}$ of the corresponding configuration. For example, according to Equation (4), the *a8-w8* and the *a8-w6* configurations in Figure 4 can perform up to 3 MAC/cycle, while the *a6-w4* configuration, featuring a *input-cluster*$_{size}$ of 4 elements, can perform up to 4 MAC/cycle. When the number of elements left in one of the *μ-vectors* is less than the *input-cluster*$_{size}$, the DSU selects a smaller chunk of elements, and reads a new *μ-vector* from the corresponding *Source Buffer*.

Data selected by the DSU are then forwarded to the Data Conversion Unit (DCU), which converts them to the appropriate *cw*, according to Equation (3). The main DCU purpose is to create the *input-clusters*, and to forward them to the 64-bit processor multiplier. The DCU also performs operand sign extensions before the actual multiplication in case of signed computations, or zero-extends each data in case the *Control Unit* flags an unsigned computation. Note that the DSU and the DCU modules apply the first two *binary segmentation* steps (colored in green and pink in Figure 1 and Figure 5, respectively).

The processor multiplier computes the *input-cluster* pair inner-product on each execution cycle, thus performing SIMD computations whose throughput ranges from 3 MAC/cycle to 7 MAC/cycle depending on the selected configuration.

The multiplication output is then filtered by the Data Filtering Unit (DFU) which, according to Equation (5), extracts the *input-clusters* inner-product, which is then accumulated into the AccMem through the internal adder. The *Control Unit* selects the suitable AccMem address among its $mr \times nr$ available slots, depending on the number of execution cycles

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on February 20,2025 at 08:11:03 UTC from IEEE Xplore. Restrictions apply.

TABLE I
Mix-GEMM OPTIMAL PARAMETERS OBTAINED IN THE DSE. NUMBER OF ELEMENTS (N); ACCMEM (AM), SOURCEBUFFERS (SB).

	MACRO-KERNEL			μ-KERNEL				*μ-engine*	
	mc	**nc**	**kc**	**mr**	**nr**	**kua**	**kub**	**AM**	**SB**
N	256	256	256	4	4	4	4	16	16

required by the loaded configuration. For example, in the *a8-w8*, *a8-w6*, and *a6-w4* configurations in Figure 4, the *Control Unit* increments the AccMem address after 12, 12, and 9 accumulations, respectively, as these represent the number of execution cycles required to compute their inner-products.

The AccMem facilitates data reuse by updating the *C* *μ-panel* elements multiple times during the *μ-kernel* execution, thus avoiding increased latency, instruction count and memory traffic. Once the whole matrix-matrix multiplication between the $mr \times kca$ *A* *μ-panel* and the $kcb \times nr$ *B* *μ-panel* has been computed by the *μ-engine*, a series of $mr \times nr$ `bs.get()` instructions collect the AccMem elements holding the *C* *μ-panel*, which are then accumulated into the output matrix *C*.

The processor treats the `bs.set()`, `bs.ip()`, and `bs.get()` as single-cycle latency instructions. Therefore, the processor does not wait for the `bs.ip()` instructions completion before moving forward with the pipeline execution. As a result, while the *μ-engine* processes the *μ-vectors*, independent memory and branch instructions can make forward progress by utilizing the address generation and branch resolution FUs. The extra cycles the *μ-engine* needs to compute the entire *μ-vectors* are partially compensated by the instructions latencies interleaving `bs.ip()` instructions, and in part alleviated by the *Source Buffers*. This paradigm allows overlapping computational and memory operations, saving a high number of execution cycles from the baseline GEMM algorithm.

Using the *binary segmentation* technique as a base pillar of the *μ-engine* allows to dynamically select the number of elements computed per cycle (*i.e.*, the $input\text{-}cluster_{size}$) depending on the computation data sizes. This feature allows higher flexibility than traditional SIMD FUs on the supported data sizes combinations, as with the proposed methodology mixed-precision data are anyhow converted to a common data size (*i.e.*, the *cw*) and computed in clusters (*i.e.*, from 3 to 7 elements per cycle).

A key strength of *Mix-GEMM* relies on its scalability. For processors hosting SIMD units, the *μ-engine* can be properly sized to sustain a higher throughput. Indeed, the *Source Buffers* can store wider *μ-vectors* (*e.g.*, based on 128-bit *load* operations), while the DSU and DCU units can select and convert a wider cluster of elements, partitioning them through all the multipliers composing the processor arithmetic FUs. Similarly, the performance benefits of *Mix-GEMM* also apply to processors hosting multiple cores. Indeed, our BLIS-based library can easily enable multi-threading support [73] while retaining performance-per-core close to the single-threaded implementation [67], and a *μ-engine* can be instantiated on every processor core with a negligible impact on area and power.

C. Design Space Exploration

As detailed in Section III-A and Section III-B, *Mix-GEMM* defines several parameters, that need to be fine-tuned to guarantee minimal overheads during the GEMM computation. Therefore, we conduct a Design Space Exploration (DSE) to select the *Mix-GEMM* parameters allowing the best trade-off between area and performance. The optimal value of the parameters obtained during the proposed DSE, considering both the *Mix-GEMM* software library and the *μ-engine*, are reported in Table I.

We follow the analytical approach proposed in [45] and detailed in Section II-C to find the best *panels* and *μ-panels* dimensions, according to the main SoC characteristics. Following [45], we find the optimal *mc*, *nc*, and *kc* values equal to 256, while the optimal *mr* and *nr* equal to 4. Accordingly, we set the *μ-engine* AccMem dimension to 16 elements, as it holds the entire *C* *μ-panel* composed of $mr \times nr$ elements.

We then analyze the memory overhead introduced by the *μ-vectors* zero-padded elements with respect to the maximum theoretical memory compression improvements (*i.e.*, from 1× to 4× for 8- and 2-bit data). Our analysis shows that the memory overhead introduced by the padded elements with *kua* and *kub* equal to 4 is 2.4% on average, considering all the supported configurations. As 4 is the maximum ratio among the data sizes supported by *Mix-GEMM* (*i.e.*8- and 2-bit), it represents the lower bound for *kua* and *kub*. Further increasing *kua* and *kub* would benefit the memory footprint, as it would increase the number of solutions requiring less zero-padded elements on each *μ-vectors* set. However, increasing *kua* and *kub* would be sub-optimal from a performance perspective. Indeed, according to Figure 2, the GEMM kernel needs to store $kua \times mr$ *A* *μ-vectors* and $kub \times nr$ *B* *μ-vectors* in the processor RF to minimize the number of load operations during the *μ-kernel* execution. As the RF leveraged by the target processor holds 32 registers, and since the optimal value for both *mr* and *nr* is equal to 4, setting *kua* and *kub* equal to 4 elements leads to an optimal RF utilization.

Another key parameter we explore is the *Source Buffers* depth, as small *Source Buffers* can fill too quickly, stalling the processor pipeline and preventing it from moving forward with the execution of subsequent instructions, while too deep *Source Buffers* could increase the *μ-engine* latency, forcing the processor to stall the `bs.get()` completion until the whole *C* *μ-panel* has been completely computed. We equip the *μ-engine* with a Performance Monitoring Unit (PMU) to collect its metrics during execution, and we benchmark GEMM tasks considering all the supported data sizes configurations, exploring *Source Buffers* depths of 8, 16, and 32 *μ-vectors*. Our analysis shows that the number of cycles where the processor is stalled because of full *Source Buffers* accounts for the 17.8%, 14.3%, and 11.2% for *Source Buffers* having depths of 8, 16, and 32 *μ-vectors*. The PMU also registered stalls due to `bs.get()` instructions only for *Source Buffers* of 32 *μ-vectors*, accounting for 2.3% of the total execution time, closing the overhead gap between *Source Buffers* holding 16 and 32 *μ-vectors*. Moreover, post-synthesis results show an area increase of the *μ-engine* of 67.6% when passing from 16 and 32 elements. For these reasons, we set the *Source Buffers* depth equal to 16 *μ-vectors*.

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on February 20,2025 at 08:11:03 UTC from IEEE Xplore. Restrictions apply.

IV. Experimental Evaluation

This section presents the experimental evaluation of *Mix-GEMM* in terms of throughput, energy efficiency, and area. We also perform an in-depth evaluation of representative quantized image classification CNNs, namely AlexNet [36], VGG-16 [66], ResNet-18 [31], Mobilenet-V1 [32], RegNet-x-400mf [78], and EfficientNet-B0 [70]. Our analysis focuses on CNNs since computer vision is a major driving task for artificial intelligence at the edge, which is the focus of this work. However, *Mix-GEMM* can be applied to all the DNNs quantized with any uniform affine quantization technique, and as such, any advancement in that area can be potentially leveraged by *Mix-GEMM*. For example, recent works [24], [65], [69] have demonstrated competitive quality of results for low mixed-precision quantization of BERT for Natural Language Processing (NLP), whose compute expansive kernels based on matrix-matrix multiplications could be accelerated exploiting *Mix-GEMM*.

The proposed experimental evaluation aims to find the best trade-offs in terms of accuracy and throughput, showing the potential of combining quantization and efficient narrow-precision inference acceleration. Indeed, the main novelty of *Mix-GEMM* is its ability to support all combinations of precisions between 8- and 2- bit, while guaranteeing performance that increase with the decrease of activations and weights bitwidths. This feature enables a new degree of freedom in deploying DNNs on edge devices. Indeed, the large number of configurations supported by *Mix-GEMM* widen the design space used to trade-off performance, memory, energy, and accuracy, which is of fundamental importance when targeting resource-constrained devices.

A. Experimental Setup

To benchmark *Mix-GEMM* in terms of performance and energy efficiency, we integrate its hardware *μ-engine* on an edge RISC-V SoC [68]. The target edge processor, implementing the RV64G instruction set, features a single-core, 7-stage, in-order, single-issue pipeline, while the memory hierarchy features L1 and L2 data caches having sizes of 32KB, and 512KB, respectively. The *Mix-GEMM* performance results have been compiled with the RISC-V GNU compiler toolchain [59] extended with the proposed custom instructions, and emulated on a Field Programmable Gate Array (FPGA) integrating the whole SoC. All the throughput results report the average performance over 10 subsequent runs. We extract area and energy consumption through the Cadence toolset, using Genus 19.11 for the synthesis and Innovus 20.1.2 for the PnR. Energy estimations have been evaluated post-PnR to have an accurate activity factor for each gate.

For QAT, we adopt PyTorch 1.8 [56], a popular deep learning framework, and Brevitas 0.7.1 [55], a neural networks quantization library. All experiments are retrained with QAT from post-training quantization of FP32 models [1], [46], which we also consider as the accuracy baseline of the target CNNs. We train on the ImageNet training dataset [19] with four NVIDIA V100 Graphics Processing Units (GPUs), reporting the best TOP-1 validation accuracy obtained for each configuration. We experiment with multiple separate precisions for activations and weights, except for the first and last layers, which are kept at 8-bit to preserve accuracy. Weights are quantized *per-channel* with *scale* computed from the *absmax* of the weight tensor [4], while activations are quantized *per-tensor* with *scale* learned in log domain [34]. Quantization *scales* and biases are left in floating-point. To simplify training, both activation and weights are trained with *zero-point* equal to zero. ResNet-18, AlexNet, MobileNet-V1, VGG-16, RegNet-x-400mf, and EfficientNet-B0 retrain with Stochastic Gradient Descent (SGD) featuring momentum of 0.9, weight decay $1e-4$, and initial learning rate of $1e-3$, $1e-4$, $1e-2$, $1e-3$ $4e-2$, and $3.2e-3$. We respectively employ 90, 90, 120, 45, 150, and 90 epochs, lowering the learning rate by 0.1 every 30, 30, 30, 15, 30, and 30 epochs, with a batch size of 256, 128, 128, 32, 128, and 64 per GPU. An exception is made for combinations of 8- and 7-bit, where models are fine-tuned for 5 epochs at the lowest learning rate they would reach in the normal training schedule, and for the EfficientNet-B0 configurations showing data sizes lower than 4-bits, that are trained employing 270 epochs. The initial activation post-training quantization is performed by averaging the 99.999 percentile of the activation absolute values for 8 batches [76], and then performing bias correction [50] for 8 more batches (except for VGG-16, where bias correction would lead to overflow). To improve convergence at low precision without overhauling the whole approach to quantization, AlexNet, ResNet-18, MobileNet-V1, RegNet-x-400mf, and EfficientNet-B0 *a4-w3* and *a3-w3* are retrained from *a4-w4* instead of FP32, with the same training settings as above except for weight decay at $5e-5$. Similarly, *a3-w2* and *a2-w2* are retrained from *a3-w3* results. For VGG-16, only *a3-w2* and *a2-w2* are handled separately, by first replacing *relu* with *relu*6 in the pretrained FP32 network, and then retraining with the settings above.

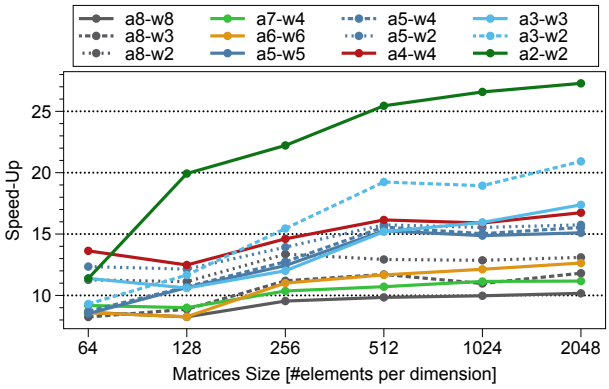

Fig. 6. Speed-up of *Mix-GEMM* over the baseline BLIS-based DGEMM algorithm on square input matrices. Configurations sharing the activations data size *(a)* are represented with the same color, with different line patterns differentiating the weights data size *(w)*.

B. Performance

We first highlight the *Mix-GEMM* scalability by analyzing its performance on general GEMM tasks, exploiting a dataset composed of square input matrices with 64 to 2048 elements per dimension. Figure 6 shows the performance increase of *Mix-GEMM* with respect to the BLIS-based DGEMM baseline, running on the same RISC-V SoC integrating the proposed *μ-engine*, for a subset of 12 activations and weights combinations. This first evaluation allows quantifying the performance benefits

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on February 20,2025 at 08:11:03 UTC from IEEE Xplore. Restrictions apply.

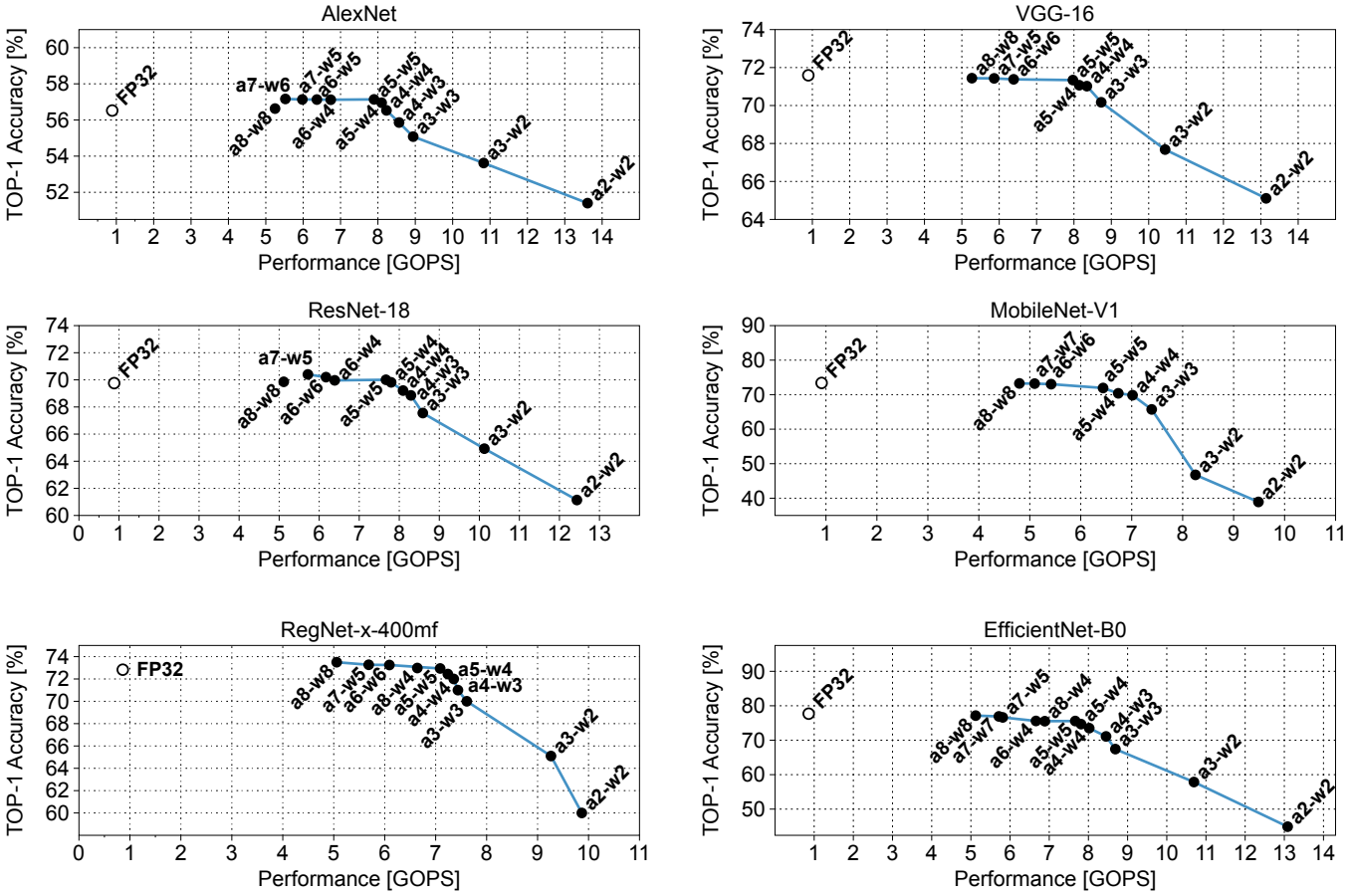

Fig. 7. Performance vs. accuracy Pareto frontier for the selected CNNs. Labels represent activations and weights data sizes (*a* and *w*), respectively. We measure the quantized networks performance exploiting *Mix-GEMM*, while the FP32 performance is measured exploiting OpenBLAS running on the SiFive U740 processor.

of the proposed hardware microarchitecture with respect to the BLIS-based DGEMM baseline. As *Mix-GEMM* keeps narrow-precision elements compressed in 64-bit data (*i.e.*, from 8 to 32 narrow-elements), it allows reducing the problem size from 8× to 32× with respect to the DGEMM implementation of BLIS. However, reducing the computation data sizes is not sufficient to guarantee high benefits in terms of performance. Indeed, BLIS running with 8-bit data only reaches an average 2.5× performance improvement with respect to the DGEMM baseline. On the other hand, as Figure 6 shows, the experimental steady-state performance of *Mix-GEMM* over the DGEMM baseline ranges from 10.2× to 27.2×, for the *a8-w8* and *a2-w2* data size configurations. Different motivations allow *Mix-GEMM* running at 8-bit to perform 10.2× and 4.1× averagely faster than the baseline running at 64- and 8-bit. First, *Mix-GEMM* keeps data compressed until the operands are issued to the *μ-engine*, thus reducing the overall number of operations, while increasing the throughput in terms of elements fetched from memory on each cycle. The *μ-engine*, computationally sustains this throughput by performing multiple MAC operations per cycle through the processor multiplier. The AccMem allows then to locally accumulate data, avoiding execution overhead due to additional *store* and *add* operations. Finally, the pipelined structure of the *μ-engine* provides a further increase in the overall throughput, as it allows to hide the `bs.ip()` operations latency without waiting for their completion. As Figure 6 also highlights, these *Mix-GEMM* benefits remain valid for any data type configuration, allowing it to actually scale its performance with the decrease of the computation data types. Specifically, the *a8-w8* performance shows a 21.6% performance improvement with respect to the theoretical lower bound of 8×, as *Mix-GEMM* exploits its AccMem to reduce the number of operations needed to update the output matrix. On the other hand, *a2-w2* shows a performance penalty of 15% with respect to the theoretical upper bound, mainly due to the high ratio between *μ-vector* size and *input-cluster*$_{size}$ (*i.e.*, 32 elements per *μ-vector* and 7 MAC/cycle of *input-cluster*$_{size}$), which implies a higher number of cycles to process the complete *μ-vector* (*i.e.*, 5 cycles). However, this overhead is only noticeable in a few configurations, and does not prevent the performance scaling of *Mix-GEMM*. Indeed, the *a4-w4* configuration in Figure 6 shows a 16× speed-up with respect to the baseline, which is in line with the theoretical one.

Aiming at evaluating *Mix-GEMM* also on SoCs tight by higher area and power constraints, we explore its performance exploiting smaller L1 and L2 caches. Our exploration, performed on all the supported data sizes and considering the same benchmark proposed in Figure 6, shows a small performance decrease when reducing the L1 cache from 64KB

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on February 20,2025 at 08:11:03 UTC from IEEE Xplore. Restrictions apply.

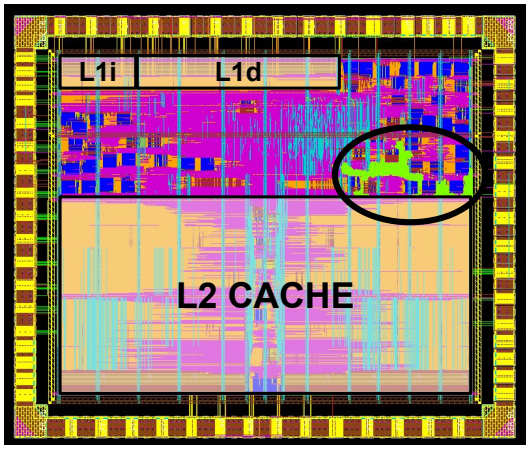

Fig. 8. Post-PnR layout, targeting the Global Foundries 22nm FDSOI technology node, of the SoC integrating the proposed *µ-engine* (highlighted in green).

to 16KB or the L2 cache from 512KB to 64KB, accounting for 5.2% and 7% on average, respectively. Decreasing both the L1 and L2 sizes (*i.e.*, 16KB and 64KB) allows reducing the SoC area by 53%, and still allows *Mix-GEMM* to achieve high performance, with an average penalty of 11.8%.

Figure 7 reports the most performant combinations of each network TOP-1 accuracy and the corresponding *Mix-GEMM* throughput, accounting for the execution time spent on each convolutional layer. We also highlight the Pareto-optimal curve representing the best trade-offs between performance and network accuracy for each network. Configurations that are not on the Pareto frontier are not reported in Figure 7, with the exception of the *a8-w8* configuration. The FP32 performance baseline has been measured with the well-known OpenBLAS library [77], exploiting single-threading and running on the SiFive U740 RISC-V processor, featuring a 64-bit dual-issue in-order pipeline running at 1.2 GHz. As Figure 7 shows, *Mix-GEMM* outperforms the FP32 baseline on all the benchmarked CNNs by a factor ranging from 5.8× to 15.1× for AlexNet, from 5.8× to 14.6× for VGG-16, from 5.7× to 13.8× for ResNet-18, from 5.3× to 10.6× for MobileNet-V1, from 5.7× to 11× for RegNet-x-400mf, and from 5.7× to 14.5× for EfficientNet-B0.

Accuracy-wise, Figure 7 shows that all the considered networks maintain a TOP-1 accuracy close to or better than the FP32 baseline for data sizes larger than 4-bit on both activations and weights, showing accuracy losses below 1.5%. This result demonstrates the benefits of the proposed solution in supporting non-standard data sizes. For example, *Mix-GEMM* can exploit the *a5-w5* configuration and reach a 60% performance improvement with respect to the *a8-w8* configuration on the selected networks, while guaranteeing similar accuracy and saving 60% in memory usage. Figure 7 also shows minimal accuracy drops, with respect to the FP32 baseline, on configurations exploiting 4-bit as minimum data size, with losses ranging from 0.01% for AlexNet, up to 4.2% on EfficientNet-B0.

For more aggressive quantizations, exploiting 3- and 2-bit data sizes, the considered networks show accuracy losses ranging from 0.5% to 5.1% for AlexNet, from 1.2% to 6.5% for VGG-16, from 2.2% to 8.6% for ResNet-18, from 7.6% to 34.5% for MobileNet-V1, from 2.6% to 13% for RegNet-x-400mf, and from 10.3% to 32.8% for EfficientNet-B0. Note that, to minimize complexity and support reproducibility, all results have been obtained by applying the same quantization techniques across all networks and data sizes, with limited hyperparameter tuning. We expect lower losses at 3- and 2-bit data sizes applying more tailored low mixed-precision techniques, such as [6], [8], [64]. Nonetheless, our results show that *Mix-GEMM* can extend the Pareto frontier to additional data sizes, capable of providing speed-up for a given accuracy target. This feature is particularly useful for edge deployment scenarios, where a trade-off between performance and quality of results typically has to be reached.

TABLE II
µ-engine AREA BREAKDOWN

Component	Area [µm^2]	SoC Overhead [%]
Src Buffers	4934.63	0.36
DSU	1094.45	0.08
DCU	2832.46	0.21
DFU	1842.25	0.13
Adder	741.58	0.05
AccMem	1214.35	0.09
Control Unit	981.43	0.08
Total: *µ-engine*	**13641.14**	**1.00**

C. Physical Design and Energy Efficiency

To present the physical layout and extract the main physical design metrics regarding area, timing, and power, we implement the RISC-V SoC including the proposed hardware *µ-engine* in the Global Foundries 22FDX 22nm FDSOI technology. We set the target frequency to 1.2 GHz for both synthesis and PnR, using 8-track standard cells without exploiting body-biasing. The SoC layout, depicted in Figure 8, features a total area of 1.96 mm^2, and includes the RISC-V in-order core, the *µ-engine* (highlighted in green and circled in black), the IO pad-ring, and the uncore composed of L2, L1d, and L1i caches of size 512KB, 32KB, and 16KB, respectively. The *µ-engine* occupies a total area of 0.014 mm^2, and adds an overhead of 1% on the total chip area. Table II highlights the area breakdown of the proposed hardware *µ-engine*, and reports the area overhead of every *µ-engine* component on the SoC. The main area contribution is given by the *Source Buffers*, implemented as 64-bit wide registers holding 16 entries. The other *µ-engine* components introduce an area overhead in the SoC smaller than 0.3%. Our synthesis and PnR evaluation evidence that the *µ-engine* does not add critical paths in the design, and introduces a post-layout estimated power consumption overhead of 2.3%. We compute the energy efficiency of *Mix-GEMM* by performing a post-PnR gate-level simulation of the SoC executing the selected CNNs, and considering the total power consumption of the *µ-engine* and the processor multiplier. Our evaluation shows that *Mix-GEMM* achieves from 522.1 GOPS/W to 1.3 TOPS/W for the computation of AlexNet, from 524.3 GOPS/W to 1.3 TOPS/W on VGG-16, from 509 GOPS/W to 1.2 TOPS/W on ResNet-18, from 477.5 GOPS/W to 944.1 GOPS/W on MobileNet-V1, from 503.3 GOPS/W to 982 GOPS/W on RegNet-x-400mf, and from 509.7 GOPS/W to 1.3 TOPS/W for EfficientNet-B0.

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on February 20,2025 at 08:11:03 UTC from IEEE Xplore. Restrictions apply.

TABLE III
COMPARISON WITH STATE-OF-THE-ART: PERFORMANCE AND EFFICIENCY RANGES ORDERED ACCORDING TO THE SUPPORTED DATA SIZES (*e.g.*, 8B – 2B). RESULTS GATHERED FROM PUBLISHED PAPERS.

	Architecture						Benchmarks													
							Convolution*		AlexNet		VGG-16		ResNet-18		MobileNet-V1		RegNet		EfficientNet-b0	
	Data sizes		SoC	Freq	Tech.	Area	Perf.	Eff.	Perf.	Eff.	Perf.	Eff.	Perf.	Eff.	Perf.	Eff.	Perf.	Eff.	Perf.	Eff.
	Supported	Mixed		[GHz]	[nm]	[mm²]	[GOPS]		[GOPS]	[TOPS/W]	[GOPS]	[TOPS/W]	[GOPS]	[TOPS/W]	[GOPS]	[TOPS/W]	[GOPS]	[TOPS/W]	[GOPS]	[TOPS/W]
Baseline	FP32	✗	RV64	1.2	-	-	-	-	0.9	-	0.9	-	0.9	-	0.9	-	0.9	-	0.9	-
[33]	8b	✗	ARMv8#	1.2	-	-	-	-	5.6	-	5.1	-	4.7	-	5.5	-	4.8	-	5.8	-
[12]	8b	✗	**8**×RV32†	0.26	-	-	-	-	-	-	-	-	-	-	4.2	0.02§	-	-	-	-
[13]	8b/4b/2b	✓	ARMv7	0.48	-	-	-	-	-	-	-	-	-	-	0.3–0.5	0.001–0.002§	-	-	-	-
[26]	8b/4b/2b	✗	RV32†	0.17	-	-	0.6–0.2	-	-	-	-	-	-	-	-	-	-	-	-	-
[11]	8b/4b/2b	✓	**8**×RV32†	0.17	-	-	6.1–2.4	-	-	-	-	-	-	-	-	-	-	-	-	-
[52]	8b/4b/2b	✓	RV32†	0.25	22	0.002 ‡	1.1–3.3	0.2–0.6	-	-	-	-	-	-	-	-	-	-	-	-
[27]	8b/4b/2b	✗	**8**×RV32†	0.6	22	0.04‡	19.8–47.9	0.7–1.1	-	-	-	-	-	-	-	-	-	-	-	-
[58]	8b/4b/2b	✗	RV64	0.6	22	0.000419	-	-	0.4–1.3	0.01–0.5§	0.6–2.5	0.01–0.03§	-	-	-	-	-	-	-	-
[17]	16b	✗	Decoupled	0.25	65	12.25	-	-	74.7	0.3	21.4	0.09	-	-	-	-	-	-	-	-
[41]	a16, w1-w16	✗	Decoupled	0.2	65	16	-	-	461.1	1.6	567.3	1.9	-	-	-	-	-	-	-	-
This work	All 8b-2b	✓	RV64	1.2	22	0.0136	4.2–7.9	0.4–0.8	5.2–13.6	0.5–1.3	5.3–13.1	0.5–1.3	5.1–12.4	0.5–1.2	4.8–9.5	0.5–0.9	5.1–9.9	0.5–1.0	5.1–13.1	0.5–1.3

* Considers an input tensor of shape ($H \times W \times F_{in}$) $16\times16\times32$, and a filter of shape ($F_{out} \times K_{dim} \times K_{dim} \times F_{in}$) $64\times3\times3\times32$
† Equipped with custom ISA extension exploiting hardware loops, post-increment load and store, and 4×8-bit MAC FUs
‡ Area only includes extension for 4- and 2-bit MAC FUs
§ Energy efficiency refers to the entire SoC
# Exploits the Neon SIMD extension

V. COMPARISON WITH STATE-OF-THE-ART SOLUTIONS

Accelerating quantized DNNs represents a widespread research topic [16], [20], [30]. Although several representative works targeting GPUs [42] and FPGAs [10], [57] are present in the literature, this section mainly considers related research works targeting CPU architectures in the edge domain. We divide the related work into three main categories proposing different approaches to optimize quantized DNNs computations on edge devices.We first consider DNN software libraries exploiting existing edge processors to efficiently compute GEMM kernels based on quantized data. We then analyze hardware-software co-designed architectures computing DNNs on edge processors adopting ISA extensions and custom FUs. We finally list the most relevant works proposing decoupled DNN accelerators for edge devices. A detailed comparison with the most relevant related works is then presented in Table III.

Optimized Software Libraries. Application-specific libraries targeting commercial edge processors, such as Facebook QNNPack [23], Arm CMSIS-NN [37], and Google GEMMLowp [33] are often used to boost the performance of quantized DNNs on edge processors. To compare the performance of *Mix-GEMM* with respect to these State-of-the-Art (SoA) software libraries, we execute the considered CNNs exploiting GEMMLowp, adopted in TensorFlow Lite [3], and highly optimized for computations based on 8-bit quantized data. The GEMMLowp benchmarks have been performed on an Arm Cortex-A53 processor, one of the most widely used architectures targeting the edge, in single-threaded mode. The Arm Cortex-A53 features a 64-bit, 8-stages, dual-issue in-order pipeline running at 1.2GHz and exploiting the NEON SIMD extension. As Table III shows, the GEMMLowp performance [33] are comparable with *Mix-GEMM* when computing the same networks considering its *a8-w8* configuration. However, as GEMMLowp does not currently support less than 8-bit based computations as a consequence of the underlying ISAs limitations, *Mix-GEMM* allows for better performance, while guaranteeing comparable accuracy. For example, from Figure 7 it can be noted that the *a5-w5* configuration of *Mix-GEMM* is capable of providing up to 70% better performance than GEMMLowp, while losing only 0.22% of accuracy on average among the selected networks.

A remarkable solution targeting the RISC-V ISA is Dory [12], a framework to deploy DNNs on the GAP-8 processor [25], reaching up to 4.2 GOPS performance to compute the convolutional layers of MobileNet-V1 at 8-bit on eight cores running in parallel. Compared to Dory, our solution achieves up to 2.6× better performance on MobileNet-V1, even running on a single core.

Although these libraries feature high performance on 8-bit computations, they do not support computations targeting sub-byte operands, as compressed data in memory need to be converted to 8-bit to exploit the SIMD operations offered by current commercial ISAs. These limitations are highlighted in CMix-NN [13], proposing an inference library for DNNs optimized for Arm processors and targeting 8-, 4-, and 2-bit mixed-precision computations. CMix-NN [13] demonstrates the benefits of supporting mixed-precision computations based on narrow-integers to compute DNNs inference, as they are able to scale their performance up to 2× in energy efficiency and 1.7× in throughput with respect to their 8-bit implementation. However, their MobileNet-V1 implementation latency is dominated by the lack of mixed-precision and sub-byte SIMD instructions at the ISA level. As a result, *Mix-GEMM* offers roughly one order of magnitude speedup on MobileNet-V1 when compared to CMix-NN.

Specialized Arithmetic Units. As off-the-shelf architectures and ISAs are inefficient in deploying quantized DNNs targeting data sizes lower than 8-bit, several works propose specialized FUs and custom ISA extensions to enable efficient narrow mixed-precision GEMM computations on edge devices. In this context, PULP-NN [26] exploits 8-bit SIMD MAC units and inner-product RISC-V based custom instructions to compute up to 16 8-bit MAC operations concurrently. PULP-NN proposes casting instructions to *pack* and *extract* vectors composed of lower data sizes (*i.e.*, 4- and 2-bit) while reusing the same SIMD MAC units. Although their experimental evaluation shows performance improvements against their 8-bit baseline, their casting instructions introduce overheads on 4- and 2-bit based com-

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on February 20,2025 at 08:11:03 UTC from IEEE Xplore. Restrictions apply.

putations, hence decreasing their performance improvement for lower bitwidths. Indeed, their performance reaches 0.6 GOPS for 8-bit computations, while it is limited to 0.2 GOPS for 2-bit data. Bruschi et al. [11] extend PULP-NN to support mixed-precision combinations for 8-, 4-, and 2-bit data sizes on an eight-cores RISC-V processor. As in PULP-NN, however, their work suffers from the same overheads on sub-byte data sizes, responsible for a 2.5× performance degradation when comparing 8- against 2-bit computations, as they also require additional *pack* and *extract* instructions. These limitations do not affect *Mix-GEMM*, which is capable of scaling its performance by 1.9× when targeting the same Convolution benchmark. Ottavi et al. [52] extend a RISC-V core with 4- and 2-bit based MAC units and custom controllers to enable narrow mixed-precision computations based on 8-, 4-, and 2-bit data. Performance-wise, *Mix-GEMM* is from 2.4× to 3.8× faster than [52], while supporting a greater number of data size combinations. A set of custom RISC-V ISA instructions and custom FUs to boost the performance of GEMM computations on edge devices are also proposed in XpulpNN [27]. Their hardware microarchitecture comprises SIMD units supporting from 4 8-bit to 16 2-bit *MAC/cycle*, but it is not supporting mixed-precision computations.

Note that the works in [11], [26], [27], [52] only consider a small convolutional kernel fitting the L1 cache as their experimental evaluation, which is not representative of real DNNs. Also, they neither provide performance results considering entire networks, nor explore how their ISA extensions can be integrated into high-performance software libraries such as BLAS, or how larger convolutional kernels introducing misses in the cache hierarchy would affect the performance of their proposal. Moreover, their baseline processor leverages on custom ISA extensions capable of introducing up to 3.1× performance improvement in the GEMM computation with respect to the standard RISC-V ISA [63]. These optimizations (*e.g.*, zero overhead hardware-loops) are orthogonal to *Mix-GEMM*, and can be implemented in the processor integrating *Mix-GEMM* to allow for a further performance improvement on DNN computations.

In *Bison-e* [58], the authors propose an area-efficient hardware microarchitecture that enables support for *binary segmentation* on RISC-V based edge processors. Although both *Mix-GEMM* and *Bison-e* exploit *binary segmentation* to reduce the arithmetic complexity of narrow-integer operations, we identify four critical weaknesses of *Bison-e* when compared to *Mix-GEMM*. First, *Bison-e* does not support mixed-precision data sizes. Second, it requires multiple instructions to compute the same *μ-vectors*, as it neither includes the input *Source Buffers*, nor the DSU. Third, it does not exploit data locality through the AccMem, with a consequent increase in the number of store operations. Finally, it does not include a tailored software library to compute dense matrix-matrix multiplications, but only defines a set of custom instructions extending the RISC-V ISA. All combined, these weaknesses allow *Mix-GEMM* to perform better than *Bison-e*, for factors ranging from 10.5× to 13× on AlexNet, and from 5.4× to 8.8× on VGG-16.

Table III also shows that most related works do not support computations based on mixed-precision data sizes that, as demonstrated in Section IV-B, are essential to enable efficient computations on the edge, as they have the potential to extend the Pareto frontier of modern deep learning models.

Decoupled DNN Accelerators. DNN decoupled accelerators represent a well-studied topic [5], [17], [41], [71], [80]. The high performance characterizing these accelerators are however counterbalanced by their lack of flexibility, as a large portion of the SoC has to be dedicated to the computation of a single kernel. Moreover, the software stack required by decoupled accelerators is typically more complex than the one proposed in *Mix-GEMM*, as they require specific offloading mechanisms and coherence managements handled at the hardware or software level. In Eyeriss [17], the authors exploit a bi-dimensional array of 16-bit processing elements and a custom multi-level hierarchical memory, optimized for both dense and sparse computations, exploiting a total area of 12.25 mm^2 in 65 nm CMOS technology. Aiming to address more aggressive quantization, UNPU [41] explores bit-serial MAC units supporting a fixed activations data size and from 16-bit to 1-bit weights data sizes.

Mix-GEMM achieves 0.2× and 0.6× in performance compared to Eyeriss on the AlexNet and VGG-16 computations, and exploits an energy efficiency comparable to UNPU when exploiting 8-bit data sizes. Moreover, leveraging on *DeepScaleTool* [61] to scale their area from 65 nm to 22 nm, we observe that *Mix-GEMM* requires 96.8× and 126.5× less area than Eyeriss and UNPU, respectively. Consequently, *Mix-GEMM* computing at 8-bit reaches a core area efficiency improvements (*i.e.*, GOPS/mm^2) ranging from 6.7× to 24× with respect to Eyeriss, and from 1.2× to 1.4× when compared to UNPU, on the computation of AlexNet and VGG-16. As such, we believe that *Mix-GEMM* represents a valid alternative to decoupled DNN accelerators targeting resource-constrained devices.

VI. Conclusion

This paper presents *Mix-GEMM*, a hardware-software co-designed architecture capable of accelerating quantized DNNs inference on resource-constrained devices. *Mix-GEMM* is capable of scaling the performance and the memory requirements of narrow-precision GEMM computations depending on the target data sizes, showing comparable or better performance than state-of-the-art GEMM libraries running on commercial processors. Our experimental evaluation shows that *Mix-GEMM* reaches from 4.8 GOPS to 13.6 GOPS on the computation of relevant CNN workloads, and up to 1.3 TOPS/W energy efficiency, while accounting for a negligible 1% of the total processor area. We believe our solution represents a step forward to fill the gap between the needs of quantized DNNs, requiring high-performance and flexibility in the data sizes involved in the computation, and edge-based architectures, demanding tight area and energy constraints.

Acknowledgements

This research was supported by the ERDF Operational Program of Catalonia 2014-2020, with a grant from the Spanish State Research Agency [PID2019-107255GB] and with DRAC project [001-P-001723], by the grant [PID2019-107255G-C21] funded by MCIN/AEI/ 10.13039/501100011033, by the Generalitat de Catalunya [2017-SGR-1328], and by Lenovo-BSC Contract-Framework (2020). The Spanish Ministry of Economy, Industry and Competitiveness has partially supported M. Doblas through an FPU fellowship [FPU20-04076] and M. Moretó through a Ramon y Cajal fellowship [RYC-2016-21104].

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on February 20,2025 at 08:11:03 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] Deep learning networks. [Online]. Available: https://github.com/osmr/imgclsmob

[2] "PyTorch Conv2d layer." [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems."

[4] A. Abdolrashidi, L. Wang, S. Agrawal, J. Malmaud, O. Rybakov, C. Leichner, and L. Lew, "Pareto-optimal quantized resnet is mostly 4-bit," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3085–3093, 2021.

[5] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Yodann: An architecture for ultralow power binary-weight cnn acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 48–60, 2018.

[6] H. Bai, M. Cao, P. Huang, and J. Shan, "Batchquant: Quantized-for-all architecture search with robust quantizer," *Advances in Neural Information Processing Systems*, vol. 34, pp. 1074–1085, 2021.

[7] J. Bai, F. Lu, and K. Zhang, "Onnx: Open neural network exchange," https://github.com/onnx/onnx, 2019.

[8] Y. Bhalgat, J. Lee, M. Nagel, T. Blankevoort, and N. Kwak, "Lsq+: Improving low-bit quantization through learnable offsets and better initialization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 696–697.

[9] D. Bini and V. Pan, "Polynomial division and its computational complexity," *Journal of Complexity*, vol. 2, no. 3, pp. 179 – 203, 1986.

[10] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[11] N. Bruschi, A. Garofalo, F. Conti, G. Tagliavini, and D. Rossi, "Enabling mixed-precision quantized neural networks in extreme-edge devices," in *Proceedings of the 17th ACM International Conference on Computing Frontiers*, ser. CF '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 217–220.

[12] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, and F. Conti, "Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus," *IEEE Transactions on Computers*, vol. 70, pp. 1253–1268, 2021.

[13] A. Capotondi, M. Rusci, M. Fariselli, and L. Benini, "Cmix-nn: Mixed low-precision cnn library for memory-constrained edge devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 5, pp. 871–875, 2020.

[14] B. W. Char, K. O. Geddes, and G. H. Gonnet, "Gcdheu: Heuristic polynomial gcd algorithm based on integer gcd computation," *Journal of Symbolic Computation*, vol. 7, no. 1, pp. 31 – 48, 1989.

[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] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[17] 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, 2017.

[18] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," 2014.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[20] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[21] L. Deng and Y. Liu, *Deep learning in natural language processing*. Springer, 2018.

[22] M. Dukhan, "The indirect convolution algorithm," *ArXiv*, vol. abs/1907.02129, 2019.

[23] M. Dukhan, Y. Wu, and H. Lu, "Qnnpack: Open source library for optimized mobile deep learning," 2018.

[24] A. Fasoli, C.-Y. Chen, M. Serrano, X. Sun, N. Wang, S. Venkataramani, G. Saon, X. Cui, B. Kingsbury, and W. Zhang, "4-bit quantization of lstm-based speech recognition models," *arXiv preprint arXiv:2108.12074*, 2021.

[25] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "Gap-8: A risc-v soc for ai at the edge of the iot," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2018, pp. 1–4.

[26] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "Pulp-nn: accelerating quantized neural networks on parallel ultra-low-power risc-v processors," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, p. 20190155, 02 2020.

[27] A. Garofalo, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, "Xpulpnn: Enabling energy efficient and flexible inference of quantized neural networks on risc-v based iot end nodes," *IEEE Transactions on Emerging Topics in Computing*, 2021.

[28] E. Georganas, S. Avancha, K. Banerjee, D. Kalamkar, G. Henry, H. Pabst, and A. Heinecke, "Anatomy of high-performance deep learning convolutions on simd architectures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. IEEE Press, 2018.

[29] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," *arXiv e-prints*, p. arXiv:2103.13630, Mar. 2021.

[30] C. Hao, J. Dotzel, J. Xiong, L. Benini, Z. Zhang, and D. Chen, "Enabling design methodologies and future trends for edge ai: Specialization and codesign," *IEEE Design & Test*, vol. 38, pp. 7–26.

[31] 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*, 2016, pp. 770–778.

[32] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 04 2017.

[33] B. Jacob and P. Warden, "gemmlowp: A small self-contained low-precision gemm library," 2022.

[34] S. Jain, A. Gural, M. Wu, and C. Dick, "Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 112–128, 2020.

[35] E. Kravchik, F. Yang, P. Kisilev, and Y. Choukroun, "Low-bit quantization of neural networks for efficient inference," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, Oct 2019.

[36] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[37] L. Lai and N. Suda, "Enabling deep learning at the iot edge," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '18. New York, NY, USA: Association for Computing Machinery, 2018.

[38] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4013–4021, 2016.

[39] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, "Basic linear algebra subprograms for fortran usage," *ACM Trans. Math. Softw.*, vol. 5, pp. 308–323, 09 1979.

[40] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[41] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "Unpu: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, 2019.

[42] G. Li, J. Xue, L. Liu, X. Wang, X. Ma, X. Dong, J. Li, and X. Feng, "Unleashing the low-precision computation potential of tensor cores on gpus," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 90–102.

[43] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu, "Brecq: Pushing the limit of post-training quantization by block reconstruction," 02 2021.

[44] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[45] T. M. Low, F. D. Igual, T. M. Smith, and E. S. Quintana-Orti, "Analytical modeling is enough for high-performance blis," *ACM Trans. Math. Softw.*, vol. 43, no. 2, aug 2016.

[46] S. Marcel and Y. Rodriguez, "Torchvision the machine-vision package of torch," in *Proceedings of the 18th ACM International Conference on Multimedia*, ser. MM '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 1485–1488.

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on February 20,2025 at 08:11:03 UTC from IEEE Xplore. Restrictions apply.

[47] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through ffts," *CoRR*, vol. abs/1312.5851, 2014.

[48] J. Meng, C. Zhuang, P. Chen, M. Wahib, B. Schmidt, X. Wang, H. Lan, D. Wu, M. Deng, Y. Wei, and S. Feng, "Automatic generation of high-performance convolution kernels on arm cpus for deep learning," *IEEE Transactions on Parallel and Distributed Systems*, no. 01, pp. 1–1, jan 5555.

[49] L. Meng and J. Brothers, "Efficient winograd convolution via integer arithmetic," *ArXiv*, vol. abs/1901.01965, 2019.

[50] M. Nagel, M. van Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1325–1334, 2019.

[51] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE access*, vol. 7, pp. 19 143–19 165, 2019.

[52] G. Ottavi, A. Garofalo, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, "A mixed-precision risc-v processor for extreme-edge dnn inference," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 512–517.

[53] V. Pan, *How to Multiply Matrices Faster*. Berlin, Heidelberg: Springer-Verlag, 1984.

[54] V. Pan, "Binary segmentation for matrix and vector operations," *Computers and Mathematics with Applications*, vol. 25, no. 3, pp. 69 – 71, 1993.

[55] A. Pappalardo, "Xilinx/brevitas," 2021. [Online]. Available: https://doi.org/10.5281/zenodo.3333552

[56] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[57] E. Reggiani, E. Del Sozzo, D. Conficconi, G. Natale, C. Moroni, and M. D. Santambrogio, "Enhancing the scalability of multi-fpga stencil computations via highly optimized hdl components," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 14, no. 3, aug 2021.

[58] E. Reggiani, C. R. Lazo, R. F. Bagué, A. Cristal, M. Olivieri, and O. S. Unsal, "Bison-e: A lightweight and high-performance accelerator for narrow integer linear algebra computing on the edge," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022, p. 56–69.

[59] RISC-V GNU Compiler Toolchain. [Online]. Available: https://github.com/riscv/riscv-gnu-toolchain

[60] RISC-V "V" Vector Extension v0.9. [Online]. Available: https://github.com/riscv/riscv-v-spec/releases/tag/0.9

[61] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[62] A. Schönhage, "Asymptotically fast algorithms for the numerical muitiplication and division of polynomials with complex coefficients," in *European Computer Algebra Conference*. Springer, 1982, pp. 3–15.

[63] F. Schuiki, F. Zaruba, T. Hoefler, and L. Benini, "Stream semantic registers: A lightweight risc-v isa extension achieving full compute utilization in single-issue cores," *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 212–227, 2020.

[69] S. A. Tailor, J. Fernandez-Marques, and N. D. Lane, "Degree-quant: Quantization-aware training for graph neural networks," in *International Conference on Learning Representations*, 2021.

[64] M. Shen, F. Liang, R. Gong, Y. Li, C. Li, C. Lin, F. Yu, J. Yan, and W. Ouyang, "Once quantization-aware training: High performance extremely low-bit architecture search," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5340–5349.

[65] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Q-bert: Hessian based ultra low precision quantization of bert," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 8815–8821.

[66] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015.

[67] T. Smith, R. van de Geijn, M. Smelyanskiy, J. Hammond, and F. Zee, "Anatomy of high-performance many-threaded matrix multiplication," 05 2014, pp. 1049–1059.

[68] V. Soria-Pardos, M. Doblas, G. López-Paradís, G. Candón, N. Rodas, X. Carril, P. Fontova-Musté, N. Leyva, S. Marco-Sola, and M. Moretó, "Sargantana: A 1 GHz+ in-order RISC-V processor with SIMD vector extensions in 22nm FD-SOI," in *25th Euromicro Conference on Digital System Design (DSD)*, 2022.

[70] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[71] T. Tang, S. Li, L. Nai, N. Jouppi, and Y. Xie, "Neurometer: An integrated power, area, and timing modeling framework for machine learning accelerators industry track paper," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 841–853.

[72] A. Trusov, E. Limonova, and S. Usilin, "Almost indirect 8-bit convolution for QNNs," in *Thirteenth International Conference on Machine Vision*, W. Osten, D. P. Nikolaev, and J. Zhou, Eds., vol. 11605, International Society for Optics and Photonics. SPIE, 2021, pp. 49 – 57.

[73] F. G. Van Zee, T. M. Smith, B. Marker, T. M. Low, R. A. V. D. Geijn, F. D. Igual, M. Smelyanskiy, X. Zhang, M. Kistler, and V. Austel, "The blis framework: Experiments in portability," *ACM Transactions on Mathematical Software (TOMS)*, vol. 42, no. 2, pp. 1–19, 2016.

[74] F. G. Van Zee and R. A. van de Geijn, "Blis: A framework for rapidly instantiating blas functionality," *ACM Trans. Math. Softw.*, vol. 41, no. 3, jun 2015.

[75] P. Wang, Q. Chen, X. He, and J. Cheng, "Towards accurate post-training network quantization via bit-split and stitching," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9847–9856.

[76] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: Principles and empirical evaluation," *ArXiv*, vol. abs/2004.09602, 2020.

[77] Z. Xianyi, W. Qian, and Z. Chothia, "Openblas," *URL: http://xianyi. github. io/OpenBLAS*, vol. 88, 2012.

[78] J. Xu, Y. Pan, X. Pan, S. Hoi, Z. Yi, and Z. Xu, "Regnet: self-regulated network for image classification," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[79] Y. Zhou, M. Yang, C. Guo, J. Leng, Y. Liang, Q. Chen, M. Guo, and Y. Zhu, "Characterizing and demystifying the implicit convolution algorithm on commercial matrix-multiplication accelerators," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2021, pp. 214–225.

[80] B. Zimmer, R. Venkatesan, Y. S. Shao, J. Clemons, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. S. Emer, C. T. Gray, S. W. Keckler, and B. Khailany, "A 0.32–128 tops, scalable multi-chip-module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 4, pp. 920–932.

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on February 20,2025 at 08:11:03 UTC from IEEE Xplore. Restrictions apply.