This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2024.3500369

Mix-GEMM: Extending RISC-V CPUs for Energy-Efficient Mixed-Precision DNN Inference using Binary Segmentation

Jordi Fornt*, Enrico Reggiani*, Pau Fontova-Musté, Narcís Rodas, Alessandro Pappalardo, Osman Sabri Unsal, Adrián Cristal, Josep Altet, Francesc Moll, and Jaume Abella

***Abstract*— Efficiently computing Deep Neural Networks (DNNs) has become a primary challenge in today's computers, especially on devices targeting mobile or edge applications. Recent progress on Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT) has shown that the key to high energy efficiency lies in executing deep learning models with low- (8- to 5-bit) or ultra-low-precision (4- to 2-bit). Unfortunately, current Central Processing Unit (CPU) architectures and Instruction Set Architectures (ISAs) present severe limitations on the range of data sizes supported to compute DNN kernels. In this work, we present *Mix-GEMM*, a hardware-software co-designed architecture that enables RISC-V processors to efficiently compute arbitrary mixed-precision DNN kernels, supporting all data size combinations from 8- to 2-bit. By applying *binary segmentation*, our architecture can scale its throughput by decreasing the data size of the operands, resulting in a flexible approach capable of leveraging state-of-the-art QAT and PTQ to achieve high energy efficiency at a very low cost. Evaluating our *Mix-GEMM* architecture in a dual-issue in-order RISC-V processor shows that we are able to boost its performance and energy efficiency by up to $44\times$ and $11\times$ with respect to the baseline processor, with an area overhead of only 2%. This allows our extended processor to execute state-of-the-art DNNs with significantly higher performance and energy efficiency than the standard FP32 precision, while retaining almost the same model accuracy.**

***Index Terms*—Deep neural networks, RISC-V extensions, energy efficiency, binary segmentation, neural accelerators**

I. Introduction

Deep Neural Networks (DNNs) have become the go-to artificial intelligence models for many applications across academia and industry. These models achieve a remarkable accuracy in a wide range of tasks, which comes at the expense of computational complexity, mainly from linear algebra operations like General Matrix Multiplications (GEMMs). Optimizing DNNs poses a challenge in many fields, particularly when tailoring them to edge and mobile hardware architectures, which demand high performance while imposing stringent constraints on area, memory and energy consumption.

Quantization is a widely used technique that enables the execution of DNNs with high efficiency by reducing the numerical precision used to represent the model parameters and intermediate values. Integer quantization, which focuses on using narrow-integer data formats (typically from 8- to 2-bit), has become very popular recently [1]. This technique improves performance and energy efficiency by reducing the memory footprint and bandwidth required by the model, as well as enabling the computation of GEMMs with low-precision hardware. DNNs are typically quantized through either Post-Training Quantization (PTQ) or Quantization-Aware Training (QAT). In PTQ methods, a pre-trained DNN model in floating-point format undergoes a calibration process to determine suitable values for *scales* and *zero-points*, used to map the values to integer format. On the other hand, QAT integrates quantization during training, which allows it to mitigate quantization-related errors. Even though retraining is very time-consuming, it allows QAT to achieve higher accuracy than PTQ with very low precisions.

Implementing quantization opens up a large design space, since there are many options to consider and degrees of freedom to exploit. For example, one can choose to have different bit-widths for the network weights and the activations, which is known as *mixed-precision computation*. Furthermore, each layer of the DNN may have a different precision, since some layers are more susceptible to quantization than others [2]. Exploiting these degrees of freedom is a promising solution for achieving very high energy-efficiency DNN execution.

However, most of the current general-purpose Central Processing Unit (CPU) architectures lack adequate support for efficiently handling narrow-precision formats, as most Instruction Set Architectures (ISAs) neither support data sizes smaller than 8-bit, nor mixed-precision computations. Although modern Single Instruction Multiple Data (SIMD) extensions and hardware accelerators are progressively enhancing support for narrow-precision data sizes and mixed-precision computations on CPUs [3], [4], [5], [6], their coverage remains limited in terms of data size granularity. Thus, exploiting fine-grained quantization of DNNs on modern processors does not consistently yield tangible performance benefits, as quantized data has to be either stored sub-optimally in memory (*i.e.*, using larger data sizes, supported by the processor ISA), or decompressed before the actual computation, requiring costly bit-manipulation operations.

In this work, we introduce *Mix-GEMM*, a novel hardware-software co-designed architecture targetting DNN applications that allows RISC-V CPUs to efficiently execute GEMM-based computations using narrow-integer mixed-precision. The

*Both first authors contributed equally to this paper.
J. Fornt, A. Cristal and F. Moll are with the Barcelona Supercomputing Center (BSC) and the Universitat Politècnica de Catalunya (UPC). E. Reggiani, P. Fontova-Musté, N. Rodas, O. Sabri Unsal and J. Abella are with the BSC. A. Pappalardo is with Advanced Micro-Devices (AMD). J. Altet is with the UPC. Author emails: {jordi.fornt, enrico.reggiani, pau.fontova, narcis.rodas, osman.unsal, adrian.cristal, jaume.abella}@bsc.es, alessand@amd.com, {francesc.moll, josep.altet}@upc.edu.

0000–0000/00$00.00 © 2021 IEEE

© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2024.3500369

hardware microarchitecture of *Mix-GEMM*, called *μ-engine*, is integrated in the execution stage of the CPU and uses the *binary segmentation* technique [7] to compute the inner-product of narrow-integer data vectors while reusing the integer multiplier already present in the processor, incurring in minimal area and power overheads while effectively enabling high-performance SIMD operations. The *Mix-GEMM* architecture is highly flexible, supporting any combination of operand precisions from 8- to 2-bits, which enables the processor to perform arbitrary mixed-precision computations with high energy efficiency.

This paper is an extension of our previous work [8], which presented the baseline architecture of *Mix-GEMM*. In this enhanced version, we add the following contributions:

- We improve the *μ-engine* architecture by applying several optimizations (explicitly detailed in Section V), which result in a performance and energy efficiency increase of up to 26% and 75%, respectively.
- To demonstrate the scalability of *Mix-GEMM*, we integrate the *μ-engine* in a dual-issue RISC-V pipeline, effectively doubling the throughput of the GEMM computation while retaining comparable energy efficiency.
- We perform an in-depth Design Space Exploration (DSE) of both the single-issue and dual-issue *Mix-GEMM*-extended RISC-V processors and compare them in terms of performance and energy efficiency.
- We benchmark the performance and energy efficiency of the dual-issue *Mix-GEMM* with eight state-of-the-art DNN models, where we achieve a throughput and energy efficiency of up to 30 GOP/s and 1.38 TOP/sW, respectively, by using narrow-integer mixed-precision, while retaining a high model accuracy.

The remainder of the paper is laid out as follows. Section II provides some background regarding *binary segmentation*. Section III details the *Mix-GEMM* architecture. Section IV presents the experimental evaluation. Section V compares *Mix-GEMM* with the most relevant related work. Finally, Section VI summarizes our conclusions.

II. Binary Segmentation

Binary segmentation [7] is a mathematical technique employed to reduce the arithmetic complexity of linear algebra computations based on narrow-integer data. By representing sets of narrow-integer values as a single, wider data value called *input-cluster*, the unmodified processor Functional Units (FUs) (e.g. scalar multipliers) can be used to perform different kinds of SIMD computations. For DNN acceleration, the most useful application of *binary segmentation* is the computation of inner-products using the processor multiplier, since these form the core computation of GEMMs.

We use the notation *μ-vector* to describe vectors of n narrow-integer values, or narrow-elements, with arbitrary bit-widths. With *binary segmentation*, the inner product of two *μ-vectors* a and b with bit-width bw_a and bw_b can be computed by packing and padding their elements into *input-clusters* and multiplying these together. Since the multiplier circuit is implicitly composed of partial product generators and adders,

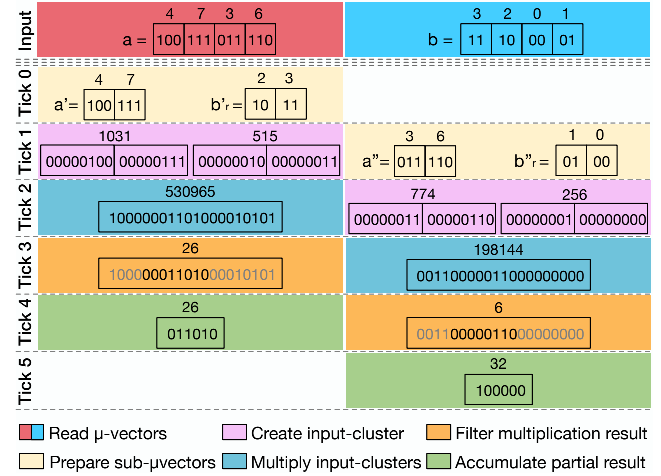


Fig. 1. Example of a pipelined inner-product computation ($4\cdot3+7\cdot2+3\cdot0+6\cdot1=32$) via *binary segmentation*. Each color represents a step required to compute the inner-product. Each tick depicts the pipeline status over time.

the idea is to arrange the elements of the a and b *μ-vectors* in such a way that the multiplication of the *input-clusters* is equivalent to the inner-product $a \cdot b$.

Two rules are used to determine the packing strategy needed to generate the correct *input-clusters* for an inner-product. The expression defined in Equation (1) determines the minimum total bit-width (including padding) of the elements packed into each *input-cluster*, called *clustering-width (cw)*.

$$cw \geq 1 + bw_a + bw_b + \lceil \log_2(\textit{input-cluster}_{size} + 1) \rceil \tag{1}$$

Where $\textit{input-cluster}_{size}$ is the number of narrow-elements that fit in the *input-cluster*, and is constrained by both the cw and the multiplier width (mul_{width}) as expressed in Equation (2):

$$\textit{input-cluster}_{size} = \left\lfloor \frac{mul_{width}}{cw} \right\rfloor \tag{2}$$

If the $\textit{input-cluster}_{size}$ is larger than 1, the computational complexity of the inner-product is reduced, since each multiplication (executed in a single clock cycle) generates the inner-product of $\textit{input-cluster}_{size}$ narrow-elements.

Figure 1 details the steps required by *binary segmentation* to compute the inner-product of two *μ-vectors* with $n=4$ elements and bit-widths $bw_a=3$, $bw_b=2$. Supposing that the example uses a 16-bit multiplier, a solution to Equation (1) and Equation (2) can be found in $cw=8$, and $\textit{input-cluster}_{size}=2$. Hence, each multiplication produces the inner-product of 2 narrow-elements, so the full *μ-vector* inner-product can be computed in 2 iterations. In Figure 1, these two iterations are shown as consecutive stages of a computational pipeline, similar to the architecture explained in Section III-C.

In the first step of Figure 1 (*Tick 0*), a and b are partitioned into *sub-μvectors* with a number of elements equal to $\textit{input-cluster}_{size}$ (i.e. 2). Additionally, the order of the elements in the second operand b is reverted (generating b_r'), according to the *binary segmentation* principles [9]. A second step (*Tick 1*) packs and pads each *sub-μvector* into the respective

© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/TC.2024.3500369

input-cluster, a single 16-bit integer that contains the narrow-elements to be computed. The third step (*Tick 2*) performs the multiplication of the *input-clusters*, which returns a 32-bit result. The multiplier output must be filtered in a subsequent step (*Tick 3*) in order to extract the correct inner-product (p_{ic}), according to the following expression:

$$p_{ic} = Mul_{out}[slice_{msb}\ ;\ slice_{lsb}] \tag{3}$$

where:

$$slice_{lsb} = (input\text{-}cluster_{size} - 1) \times cw \tag{4}$$

$$slice_{msb} = slice_{lsb} + cw - 1 \tag{5}$$

Finally (*Tick 4*), the partial results are accumulated to obtain the final inner-product (*i.e.*, 32). In this example, the inner-product of 4 elements is computed via *binary segmentation* with only two 16-bit multiplications and one addition. When applied to 64-bit architectures, *binary segmentation* allows the multiplier to compute inner-products with a throughput ranging from 3 Multiply-Accumulate (MAC)/cycle to 7 MAC/cycle, for data sizes of 8- and 2-bit, respectively.

III. MIX-GEMM HW-SW ARCHITECTURE

This section is structured as follows. First, Section III-A defines the set of specific instructions we added to the RISC-V ISA. Second, Section III-B details our proposed software library, based on the BLAS-like Library Instantiation Software (BLIS), that allows to compute fast GEMMs based on the underlying hardware. Section III-C describes the *μ-engine*, our hardware microarchitecture that enables the processor to efficiently compute GEMMs with arbitrary mixed-precision by exploiting the *binary segmentation* technique. Finally, Section III-D details how we extend the *μ-engine* architecture to a dual-issue processor pipeline with two multipliers, to demonstrate the scalability of *Mix-GEMM* when more hardware resources are available.

A. Mix-GEMM RISC-V Extension

We extend the RISC-V ISA with four R-type instructions, named `set()`, `put_a()`, `put_b()`, and `get()`.

The `mxg.set()` instruction is issued to the *μ-engine* once for the entire GEMM computation to configure the unit with details about the incoming *μ-vectors*, such as the data sizes, encoding (*i.e.*, signed or unsigned) and number of vectors to reduce; as well as specify *binary segmentation* related constraints, such as the $input\text{-}cluster_{size}$, the *cw* and the slice of data to extract from the multiplication output.

The `mxg.put_a()` and `mxg.put_b()` instructions provide input data to the *μ-engine*. Each of these instructions can write up to 2 *μ-vectors* in a single clock cycle. The hardware starts a computation as soon as it receives enough data from both operands. If any of the *μ-engine* input memories is full and, simultaneously, a `mxg.put_x()` of the corresponding operand `x` is issued (where `x` can be `a` or `b`), the processor pipeline is stalled until new inputs can be accepted.

The `mxg.get()` instruction is used to extract the resulting partial products from the *μ-engine*. The results are read sequentially in the order they are computed. If a `mxg.get()` is issued when there is no data ready for extraction, the processor pipeline is stalled until the next value is ready. Possible deadlocks between `mxg.get()` and `mxg.put_x()` are prevented at the software level (see Section III-B).

All `mxg` instructions are treated as single-cycle latency instructions, so the processor can continue the application execution while the *μ-engine* is computing. This allows the processor to issue independent memory and branch instructions to, for example, load the next weight and activation data while the *μ-engine* is computing. The overlap of computational and memory operations can significantly reduce the number of execution cycles from the baseline GEMM algorithm.

B. Mix-GEMM Software Library

We build our narrow-precision software library on top of the Double-precision General Matrix Multiplication (DGEMM) kernel of BLIS [10], a state-of-the-art framework for high-performance Basic Linear Algebra Subprogram (BLAS) computations. The BLIS DGEMM kernel considers a block-based multiplication between two matrices *A* and *B*, with sizes $m \times k$, and $k \times n$, respectively. The result, matrix *C*, has dimensions $m \times n$. BLIS partitions the matrices into *panels*, stored contiguously in memory arrays. Each panel is then partitioned into several *μ-panels*, which are used by the so-called *μ-kernel* to implement the matrix-matrix multiplication operation. Hence, at the core of BLIS lies the GEMM operation between the *A*- and *B*-*μ-panels*.

The *Mix-GEMM* Software Library leverages BLIS to efficiently move the narrow-precision data through the cache hierarchy. The structure of *panels* and *μ-panels* is maintained, but instead of 64-bit double precision numbers, each element of the data structures is a *μ-vector* of narrow-integers, compressed over the k (channels) dimension. *Mix-GEMM* supports data sizes ranging from 8 to 2-bit, so each *μ-vector* can contain from 8 to 32 elements. For bit-widths that are not integer divisors of 64, the unused bits are set to zero (e.g. when using 3-bits, each *μ-vector* packs 21 elements plus one zero padding bit). This strategy allows handling arbitrary non-standard data sizes efficiently without having to extend the processor ISA.

Figure 2 shows the data structures used in the *Mix-GEMM* Software Library. 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 Register File (RF). Moreover, the *μ-engine* implements a set of Scratchpads (SPs) that store the entire *C*-*μ-panel* and part of the *A*-, *B*-*μ-panels*. This makes the memory accesses faster and less power-consuming, and allows the *μ-engine* to implement part of the GEMM loops in hardware, eliminating unnecessary transactions.

Note that, since *Mix-GEMM* supports arbitrary mixed-precision, each operand can have a different number of *μ-vectors*, as the amount of compressed narrow-elements that fit in one *μ-vector* depends on the bit-width, and the total number of narrow-elements in both operands must be equal in the *k*-dimension. Hence, we need to consider two separate *k*-dimensions for the *A* and *B* data structures.

© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2024.3500369

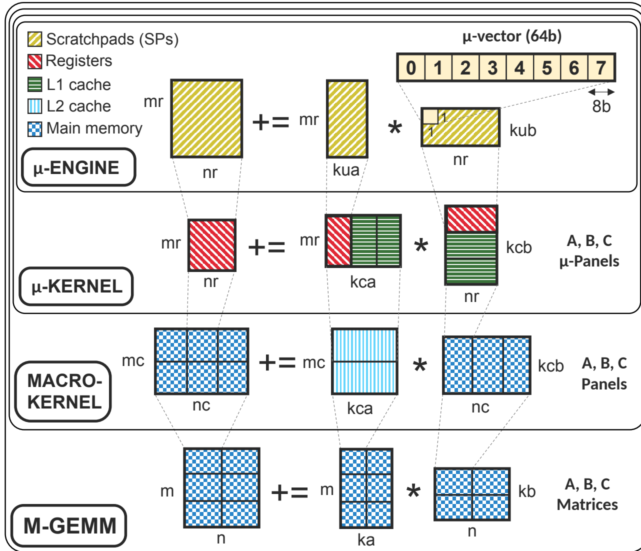


Fig. 2. Data flow, notation and allocation of the proposed MACRO-KERNEL and µ-KERNEL procedures, built upon the BLIS implementation of DGEMM. The example shows a *µ-vector* with 8-bit narrow-elements.

Algorithm 1 shows the pseudo-code of the proposed BLIS-based library implementation, whose top-function is represented by the M-GEMM procedure. M-GEMM loads the current *µ-engine* configuration through a `mxg.set()` (line 48), and then splits the *B* and *A* input matrices in *panels* (lines 51 and 53). The MACRO-KERNEL procedure further splits the *panels* into *µ-panels* (lines 39 and 41) and calls the µ-KERNEL.

Each innermost iteration of the µ-KERNEL loads $mr \times kua$ *µ-vector* pairs from the A *µ-panels*, and $nr \times kub$ pairs from the B *µ-panels*, forwarding them to the *µ-engine* through a series of `mxg.put_a()` and `mxg.put_b()` instructions. As the branches listed in the µ-KERNEL can be easily resolved by the compiler, they do not require adding branch operations at runtime. Hence, the µ-KERNEL execution is mostly composed by a series of `load()` and `mxg.put_x()` instructions. After all the elements in the current *µ-panel* have been passed to the *µ-engine*, the results are collected via $mr \times nr$ `mxg.get()` instructions (lines 30 to 34), and accumulated in the output matrix *C* (line 35).

Algorithm 1 Mix-GEMM pseudo-algorithm.

1: **procedure** µ-KERNEL($A_{\mu p}, B_{\mu p}, C_{\mu p}$)
2: **for** kca/kua $iterations$ **do** ▷ Same as kcb/kub
3: **for** $i = 0 \to mr - 1$ **do**
4: **for** $j = 0 \to kua/2$ **do**
5: $curr_addr = i + 2 * j * mr_{max}$
6: $next_addr = i + (2 * j + 1) * mr_{max}$
7: **if** $kua > 1$ **then**
8: mxg.put_a($A[curr_addr], A[next_addr]$)
9: **end if**
10: **end for**
11: **if** $kua\%2! = 0$ **then**
12: mxg.put_a($A[i + (kua - 1) * mr_{max}], 0$)
13: **end if**
14: **end for**
15: **for** $i = 0 \to nr - 1$ **do**
16: **for** $j = 0 \to kub/2$ **do**
17: $curr_addr = i + 2 * j * nr_{max}$
18: $next_addr = i + (2 * j + 1) * nr_{max}$
19: **if** $kub > 1$ **then**
20: mxg.put_b($B[curr_addr], B[next_addr]$)
21: **end if**
22: **end for**
23: **if** $kub\%2! = 0$ **then**
24: mxg.put_b($B[i + (kub - 1) * nr_{max}], 0$)
25: **end if**
26: **end for**
27: $LoadNextAddress(A_{\mu p})$ ▷ Next $kua \times mr$ elements
28: $LoadNextAddress(B_{\mu p})$ ▷ Next $kub \times nr$ elements
29: **end for**
30: **for** $i = 0 \to nr - 1$ **do** ▷ Get output from AccMem
31: **for** $j = 0 \to mr - 1$ **do**
32: $C_{\mu k}[i, j] =$ mxg.get($j + i * mr$)
33: **end for**
34: **end for**
35: $UpdateC(C_{\mu k}, C_{\mu p})$
36: **end procedure**
37: **procedure** MACRO-KERNEL(A_p, B_p, C_p)
38: **for** nc/nr $iterations$ **do**
39: $B_{\mu p} = Create\mu Panel(B_p)$
40: **for** mc/mr $iterations$ **do**
41: $A_{\mu p} = Create\mu Panel(A_p)$
42: $C_{\mu p} = Create\mu Panel(C_p)$
43: µ-KERNEL($A_{\mu p}, B_{\mu p}, C_{\mu p}$)
44: **end for**
45: **end for**
46: **end procedure**
47: **procedure** M-GEMM(A, B, C)
48: mxg.set(cfg) ▷ Load configuration
49: **for** n/nc $iterations$ **do**
50: **for** ka/kca $iterations$ **do** ▷ Same as kb/kcb
51: $B_p = CreatePanel(B)$
52: **for** m/mc $iterations$ **do**
53: $A_p = CreatePanel(A)$
54: $C_p = CreatePanel(C)$
55: MACRO-KERNEL(A_p, B_p, C_p)
56: **end for**
57: **end for**
58: **end for**
59: **end procedure**

C. µ-engine Hardware Architecture

The *µ-engine* architecture, depicted in Figure 3, is composed of 6 major blocks. Two scratchpads (*SP A* and *SP B*) hold the input operands during a computation and feed them to subsequent submodules in the correct order to compute the matrix multiplication. The *Data Selection Unit* (DSU) reads from the scratchpads and converts the incoming *µ-vectors* into the *input-clusters* that will be fed to the processor multiplier to compute the inner product via *binary segmentation*. The *Data Filtering Unit* (DFU) filters the result of the multiplication to obtain the final inner product. The accumulator scratchpad (*Acc SP*) stores the accumulated partial sums until the full matrix multiplication is finished and the results can be retreived. Finally, the *Control Unit* features a set of configurable registers and some logic that regulates the whole computation.

The *Control Unit* is fully configured by a single `mxg.set()` instruction. Hence, the overhead of reconfiguring the *µ-engine* is negligible in the computation, with respect to the complete GEMM of the *µ-panels*. Therefore, 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 to select the best trade-off between performance and accuracy.

The input scratchpads (*SP A* and *SP B*) are filled by `mxg.put_a()` and `mxg.put_b()` instructions, each writing up to 2 *µ-vectors* per cycle (see lines 8 and 12 of Algorithm 1). The scratchpad loading process is performed in sequential order (i.e. in the example on Figure 4, from [0,0] to [3,3]). As soon as both input scratchpads are non-empty, the computation starts. *SP A* and *SP B* include control logic

© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2024.3500369

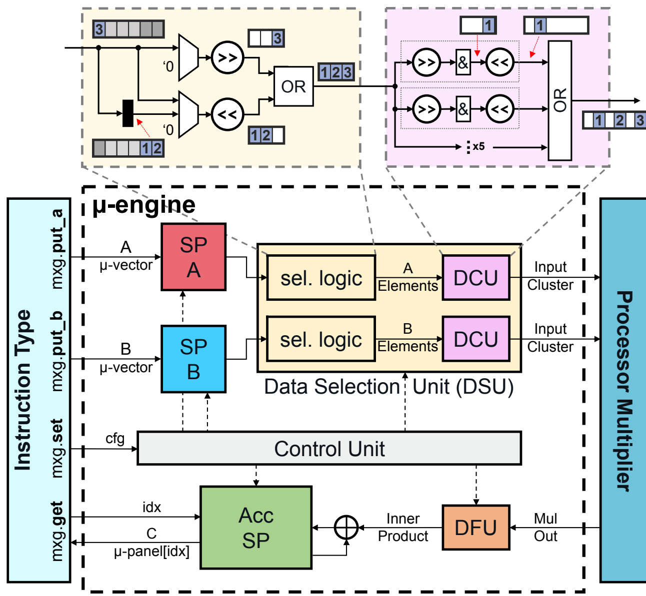


Fig. 3. *µ-engine* architecture, integrated in the processor FU. Different colors represent different stages of the *binary segmentation* technique, using the same scheme as Figure 1.

to enforce the computation of a matrix-matrix multiplication by implementing hardware loops. When each of the input scratchpads is read, its internal read pointer is updated as depicted in Figure 4. Both pointers advance first on the dimension of the reduction (kua and kub). When *SP B* has reached the last index on the kub dimension, the read pointer is decremented by $kub - 1$ positions. *SP A*, on the other hand, advances its read pointer until all the *µ-vectors* are read. When it reaches the last element, *SP A* resets its pointer value to the first element, and the read pointer on *SP B* is incremented by 1 instead of decremented (see orange arrows in Figure 4).

All memory positions of the input scratchpads are freed as soon as the values are not needed. For example, positions [0,0] to [0,3] of *SP B* on Figure 4 can be freed as soon as the read pointer advances to [1,0], as they will not be used again. Similarly, the positions of *SP A* are freed during the last iteration of the hardware loop. This enables the processor to complete the first `mxg.put_x()` instructions of the next context before the current computation is finished. The memory blocks on *SP A* and *SP B* are implemented using latches to minimize their impact on area and power.

The *DSU* features a block of selection logic that takes the appropriate narrow-elements (*i.e.*, the *sub-µvectors*) for the current input-cluster. Figure 5 shows three examples of *DSU* activity, with the *µ-vectors* representing the output of the scratchpads and the *sub-µvectors* (highlighted in different colors) being the output of the *DSU* selection logic. Note that in the examples, when the bit-width is different, the total number of *µ-vectors* needed for each operand changes. Likewise, the number of *sub-µvectors* that are computed per cycle is also different, according to the constraints of *binary segmentation* (see Section II). For example, the *a8-w8* and *a8-w6* configurations in Figure 5 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. In order

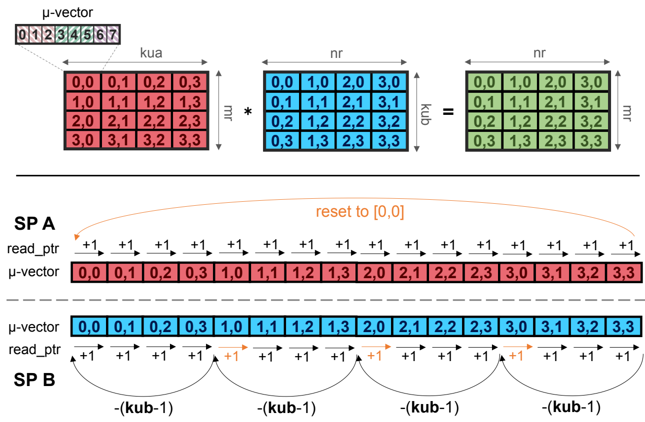


Fig. 4. Representation of the hardware loops implemented in the input SPs. In the example, kua=kub=mr=nr=4, and the bit-width is $a8$-$w8$.

to concatenate the elements from consecutive *µ-vectors*, the *DSU* includes a register to temporarily hold the previous *µ-vector* and two barrel-shifters that reshape the sub-elements accordingly, as depicted in Figure 3.

The resulting vector of selected narrow-elements is then forwarded to the *Data Conversion Unit* (DCU), which creates the *input-clusters* by introducing the necessary padding (see Figure 1). The *DCU* also performs operand sign extension in the case of signed computations. This padding and reshaping operation is also implemented using barrel shifters. In short, the selection part of the *DSU* and the *DCU* implement the first two steps of the inner product using binary segmentation (highlighted in yellow and pink in Figures 1 and 3).

The processor multiplier computes the product of the selected *input-cluster* pair on each execution cycle, thus performing SIMD computations with throughput ranging 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 extracts the inner-product of the *input-clusters* according to Equation (3). This result is accumulated into the *Acc SP* through an internal adder.

The *Acc SP* also includes some control logic that enforces the correct handling of its $mr \times nr$ available slots, depending on the number of execution cycles required for a full accumulation. For example, in the *a8-w8*, *a8-w6*, and *a6-w4* configurations in Figure 5, the write address of the scratchpad is incremented after 11, 10, and 8 accumulation cycles, respectively, which is the number of cycles needed for a complete reduction. The memory block holding the accumulated values is also implemented using latches to optimize area and power.

The partial sums stored in the *Acc SP* are kept stationary during subsequent *µ-engine* executions, since the *Mix-GEMM* software library iterates over the kca and kcb dimensions first (see Algorithm 1). In other words, the contents of the *Acc SP* are extracted only once after the whole *µ-kernel* is computed. This data reuse helps decrease latency, instruction count and memory traffic. When the *µ-kernel* computation is completed, a sequence of $mr \times nr$ `mxg.get()` instructions collect the *Acc SP* elements holding the *C µ-panel*, which are then accumulated into the output matrix *C*.

Using *binary segmentation* as the base pillar of the *µ-*

© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2024.3500369

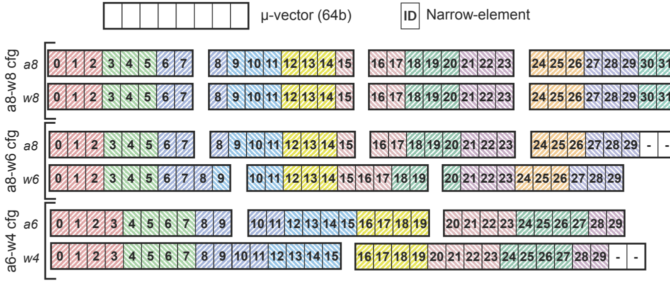


Fig. 5. Sequence of *μ-vectors* for three bit-width configurations. Different colors represent different *sub-μvectors* being computed during one cycle. In the notation aX-wY, X and Y are the activation and weight bit-widths.

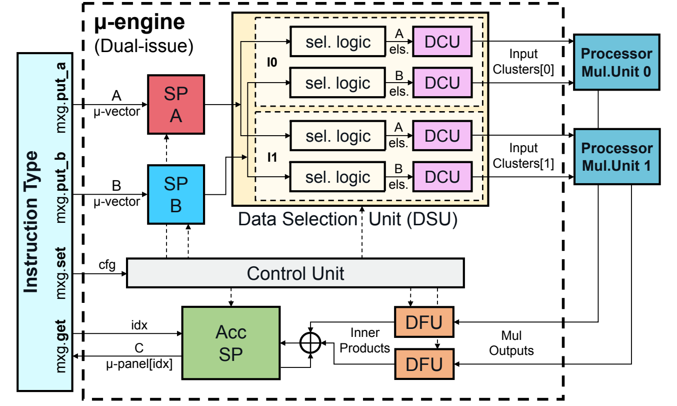


Fig. 6. Dual-issue extension of the *μ-engine* architecture. The color scheme from Figures 1 and 3 is maintained.

engine allows our hardware unit to dynamically change the number of elements computed per cycle (*i.e.*, the *input-cluster$_{size}$*) depending on the bit-width configuration, while reusing the same unmodified multiplier. This feature provides higher flexibility than traditional SIMD FUs on the supported data sizes, as arbitrary mixed-precision *μ-vectors* are converted to a common data size (*i.e.*, the *cw*) and computed in clusters (*i.e.*, from 3 to 7 elements per cycle).

D. Dual-issue μ-engine Architecture

We extend the *μ-engine* architecture to work with a dual-issue RISC-V architecture. Besides the typical extensions to the single-issue pipeline, we also include a second integer multiplier unit in the dual-issue processor, which will be used to effectively double the peak throughput of the *μ-engine*. Figure 6 depicts the extended micro-architecture, with access to both multiplier units of the dual-issue processor.

The dual-issue extension of *Mix-GEMM* serves three purposes. Firstly, it exploits the superscalar nature of the processor to improve the overall performance of our technique. Secondly, it doubles the maximum computations per cycle that the *binary segmentation* technique allows, by leveraging the two available multipliers. Lastly, it demonstrates the scalability and flexibility of our approach: with some minor changes, the *μ-engine* can be adapted to utilize the hardware resources available in the processor (in this case, the multipliers) to further accelerate mixed-precision GEMM workloads.

The modifications applied to the *μ-engine* are as follows. First, the *DSU* is extended to support two streams of outgoing *input-clusters*, by replicating the selection logic and the *DCU*s. Second, the *DFU* is replicated to filter the output of both multipliers, and the results are added before being accumulated into the *Acc SP*. The rest of the *μ-engine* blocks (namely the scratchpads and the *Control Unit*) are shared.

Even though the dual-issue processor supports executing two concurrent `mxg` instructions, the *μ-engine* only accepts one input instruction per cycle. If two `mxg` of any type are issued simultaneously, the processor is stalled during one clock cycle and the instructions are consumed one at a time. There are two reasons behind this choice: first, the main bottleneck of the *μ-engine* is the computation and not the data movement, so the potential impact of supporting concurrent dual-access in the scratchpads does not outweigh the increase in hardware complexity it entails. Second, the `mxg.put_x` and `mxg.get` instructions are typically interleaved with load and store operations, therefore supporting concurrent `mxg` instructions would have no real benefit in most cases. Due to this fact, the input throughput of the extended *μ-engine* in the context of the dual-issue RISC-V processor is effectively doubled with respect to the single-issue version, without the need for supporting concurrent instructions, as the groups of `load` + `mxg.put_x` and `mxg.get` + `store` instructions are issued together in a single clock cycle instead of two.

IV. Experimental Evaluation

To evaluate the *Mix-GEMM* architecture, we integrate the *μ-engine* on the Sargantana RISC-V processor [11]. Sargantana features a single-core, 7-stage, single-issue in-order pipeline. It implements the RV64G instruction set, which we extend by adding the *Mix-GEMM* instructions detailed in section III-A. The processor's memory hierarchy features a 512 KB L2 cache, a 16 KB instruction cache, and a 32 KB data cache. We also integrate the dual-issue *μ-engine* on a dual-issue version of Sargantana with the same memory hierarchy and stages, including two integer multipliers.

Our experimental methodology is as follows: on the one hand, we emulate the extended Sargantana single- and dual-issue systems, including the *μ-engine*, in a Xilinx Alveo U55C Field Programmable Gate Array (FPGA) to evaluate *Mix-GEMM* with different benchmarks, since RTL-level and Gate-Level Simulation (GLS) are too time-consuming. Using this setup, we perform a DSE of the architecture to find the optimal values of its parameters (Section IV-A), as well as a performance analysis using square matrices of different sizes (Section IV-B). On the other hand, we perform a full physical design of the single- and dual-issue processors in 22 nm technology, to accurately characterize the impact of the *μ-engine* in terms of area, timing and power consumption (Section IV-C). A detailed power and energy efficiency analysis is presented in Section IV-D. Finally, Section IV-E evaluates the performance and energy efficiency of *Mix-GEMM* for DNN acceleration by analyzing the execution of different state-of-the-art CNNs with the dual-issue architecture.

© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/TC.2024.3500369

TABLE I
Mix-GEMM OPTIMAL PARAMETERS OBTAINED IN THE DSE, FOR THE SINGLE-ISSUE (*S-i*) AND DUAL-ISSUE (*D-i*) IMPLEMENTATIONS.

	MACRO-KERNEL			μ-KERNEL			*μ-engine* SPs		
	mc	nc	kc	mr	nr	ku	A	B	Acc
S-i	32	512	64	4	4	4	128 B	128 B	64 B
D-i	32	256	64	8	8	8	512 B	512 B	256 B

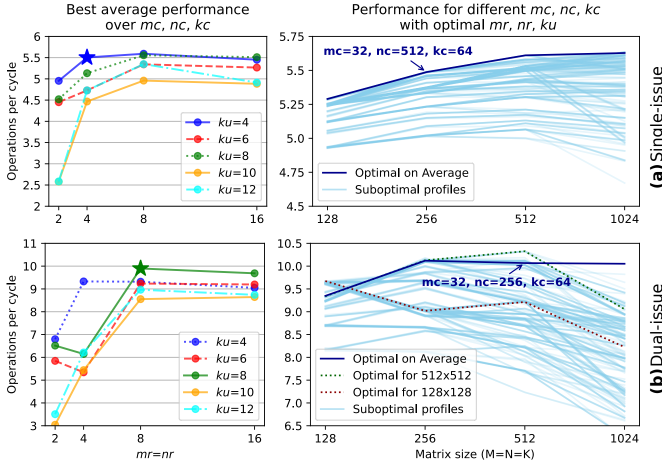


Fig. 7. DSE performance analysis results. Left: first DSE phase, points represent the maximum throughput over all $\{mc, nc, kc\}$. Right: second phase, lines represent performance profiles using the optimal $\{mr, nr, ku\}$ for different $\{mc, nc, kc\}$ combinations.

A. Design Space Exploration

The *Mix-GEMM* HW-SW architecture introduces a set of parameters that define how the GEMM matrices are tiled, namely $\{mr, nr, ku, mc, nc, kc\}$, as detailed in Section III-B (in this notation, ku is the upper limit for kua and kub). These parameters can be set freely, and their values determine how the data structures are moved through the memory hierarchy, which impacts memory stalls during computation, and therefore affects the performance and utilization of the *μ-engine*.

We tune these parameters experimentally by executing GEMM operations with different values and analyzing how the processor's performance is affected. Each parameter combination is evaluated with square matrices of size $\{128, 256, 512, 1024\}$. We perform this analysis using the $a8$-$w8$ configuration (i.e. 8-bit data in both operands) because it requires the maximum memory bandwidth, and therefore the performance of *Mix-GEMM* is more sensitive to cache-related stalls, which are affected by the DSE parameters.

The *Mix-GEMM* parameters are tuned in two phases. First, we set the parameters $\{mr, nr, ku\}$, as they impact the size of the scratchpads on the *μ-engine*. To abstract the other parameters from this analysis, we consider only the maximum performance achieved over them. The left side of Figure 7 illustrates the results of this step. The single-issue version has optimal performance at mr=nr=8 and ku=4, but as the design point mr=nr=ku=4 requires ×4 less memory and is only 1.5% slower, we choose the latter.

In the second phase, after $\{mr, nr, ku\}$ are fixed, we analyze $\{mc, nc, kc\}$, which only affect how the software library partitions the matrices. These experiments (see the right side of Figure 7) result in the optimal parameters reported in Table I. The characterization of *Mix-GEMM* presented in the next sections uses these parameters.

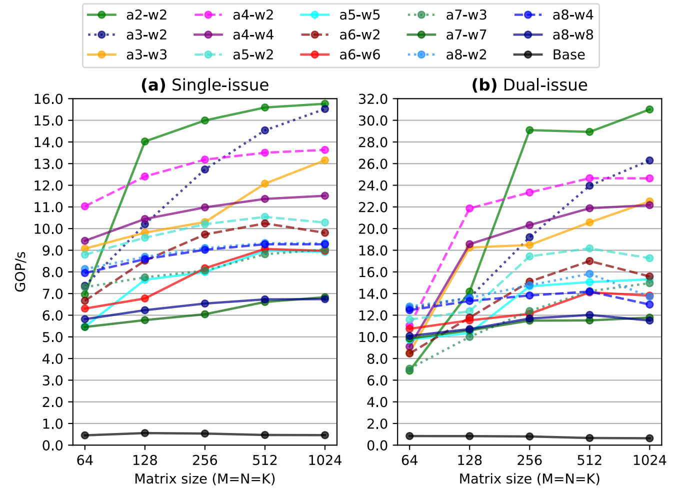


Fig. 8. Throughput of *Mix-GEMM* on square input matrices of different size. *Base* denotes the baseline BLIS-based GEMM excluding our *μ-engine*.

B. Performance Analysis

To analyze the performance of the *Mix-GEMM* architecture, we use the same FPGA emulation platform as in Section IV-A to execute matrix-matrix multiplications of different sizes with our RISC-V extension, for all the supported bit-width combinations. The emulated hardware runs at 50 MHz due to the limitations of the FPGA. It is worth noting that the *μ-engine* is not in the critical path, so our solution is not the limiting factor in terms of timing. We extrapolate the emulated results to the performance of an ASIC implementation by scaling the throughput according to the maximum frequency values obtained in the physical design (see Section IV-C).

We model and emulate the processor up to the L2 cache, and we assume that transfers between the cache and the off-chip memory will always have enough bandwidth (since the FPGA memory is faster than the emulated core). This simplified setup is a good enough approximation of a real system because the bandwidth required by the L2 (ranging from 200 MB/s to 600 MB/s when running at 1.2 GHz) is orders of magnitude below the speed of typical LPDDR memories (e.g. LPDDR4 typically reaches 17 GB/s [12]).

Figure 8 shows the throughput of the single- and dual-issue versions of *Mix-GEMM* using a subset of bit-width configurations and random square matrices of different size. The throughput of the Sargantana processor without using *Mix-GEMM*, i.e. the original BLIS (excluding our *μ-engine* and custom SW library), is denoted as *Base*. Since the baseline processor does not support narrow-integer compression, *Base* is equivalent to using an $a64$-$w64$ configuration. Our proposed *Mix-GEMM* extension is able to speed up the GEMM computation of the single- and dual-issue processors by up to $34\times$ and $44\times$, respectively.

© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2024.3500369

TABLE II
PHYSICAL DESIGN SETUP AND *μ-engine* HARDWARE SPECIFICATIONS.

Technology	GF22FDX (22 nm)		
Floorplan Area	1.54 mm x 1.26 mm		
	Typ. corner	**Fast corner**	
Voltage	0.80 V	0.88 V	
Temperature	25ºC	-40ºC	
Frequency	1.0 GHz	1.2 GHz	
	S-issue	**D-issue**	**Diff.**
μ-engine Area	10695 μm^2	30091 μm^2	2.8×
μ-engine Area overhead	0.82%	2.11%	-
μ-engine Power*	3.6 mW	8.6 mW	2.4×
Multiplier Power*	5.7 mW	10.5 mW	1.8×
Peak GOP/s	15.8	31.0	1.9×
Typ. GOP/sW*	1405	1349	-4%

* Power and energy efficiency using the typical corner with $a2$-$w2$.

In terms of operations per second, the single-issue *μ-engine* can reach from 6.7 GOP/s to 15.8 GOP/s, with $a8$-$w8$ and $a2$-$w2$, respectively, while the dual-issue version reaches from 11.5 GOP/s to 31.0 GOP/s. The upper bound of *binary segmentation* (using a 64-bit multiplier at 1.2 GHz) is 7.2 GOP/s with $a8$-$w8$ and 16.8 GOP/s with $a2$-$w2$, and twice as much with two multipliers. With that into consideration, the performance that our *μ-engine* achieves when computing large matrices is within 93% of the theoretical maximum.

Through careful examination of the performance tests, we have detected two main causes that separate *Mix-GEMM* from the theoretical upper bound. First, packing the narrow-elements in 64-bit *μ-vectors* can lead to suboptimal utilization. On one hand, some configurations can not use all the bits in the *μ-vector* itself. For example, when using 6-bit narrow integers, only 60 of the 64-bits of the *μ-vector* are used. This results in some under-utilization of memory resources. On the other hand, computing the inner-product of *μ-vectors* with different bit-width often requires padding some of the 64-bit vectors with zeros (see Figure 5) to ensure that the total elements being multiplied is the same, regardless of bit-width.

Second, cache and Translation Lookaside Buffer (TLB) misses on the memory hierarchy can cause stalls in the computation. Even though our software library is designed to maximize locality and minimize cache misses, these can appear occasionally. Stalls due to cache misses are mitigated in the *μ-engine* thanks to the hardware being able to compute independently as long as it has enough data in its scratchpads. However, cache misses when filling up the SPs can make the *μ-engine* stall until the miss is resolved.

C. *μ-engine* Hardware Characterization

We use the Cadence Electronic Design Automation (EDA) tools to implement the processors in 22 nm technology. In particular, we use Genus for low-power synthesis, Innovus for Place and Route (PnR) and Xcelium for post-layout GLS.

The final physical layout of the Sargantana processor, including our dual-issue *μ-engine*, is shown in Figure 9. The constraints and setting used for the multi-corner Synthesis and PnR are summarized in Table II, as well as the main specification of the *μ-engine* versions. According to the post-layout

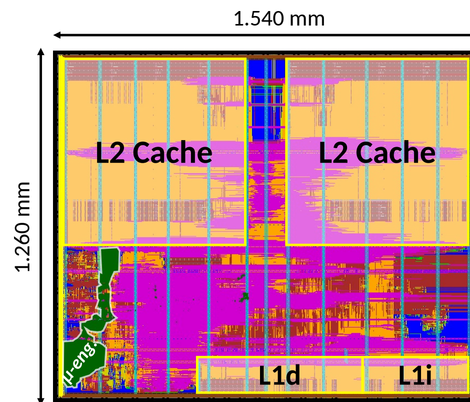


Fig. 9. Physical layout of the dual-issue Sargantana RISC-V processor integrating our *μ-engine* (cells and connections highlighted in green).

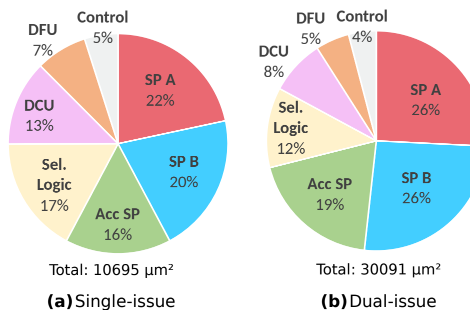


Fig. 10. Area breakdown for the single- and dual-issue *μ-engine*s.

timing reports of the single- and dual-issue implementations, the *μ-engine* is not in the critical path of the processor, so the module itself could be clocked at even higher frequencies.

The area overhead of our hardware module is only 0.82% and 2.11% of the total processor area, for the single- and dual-issue pipelines respectively. The dual-issue *μ-engine* occupies about 2.8× as much area as the single-issue version. Most of this difference can be explained by the memory requirements, since according to our DSE (see Section IV-A) the dual-issue *μ-engine* needs 1.25 KB of scratchpad memory for optimal performance, while the single-issue version only needs 0.31 KB. Figure 10 shows clearly that the dual-issue *μ-engine* devotes a much higher percentage of area to its memories. Synthesizing the dual-issue module with the same memory sizes as the single-issue version (i.e. considering $mr = nr = ku = 4$) results in an area of 15208 μm (1.42× compared to the single-issue), which proves that our hardware architecture itself scales well. Nevertheless, from a system point of view, the dual-issue *μ-engine* with 1.25 KB still represents a very tiny portion of the processor, so the area costs of our solution are small in either case, and larger memories improve the overall performance in the dual-issue case.

We estimate the power consumption of the *μ-engine* by performing GLS of the post-layout netlist and recording the switching activity of the circuit during the execution of a matrix-matrix multiplication with square matrices of size 1024. With this activity profile, we use Cadence Joules to obtain the average power consumption during computation. Section IV-D presents a detailed analysis of power and energy consumption.

© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/TC.2024.3500369

TABLE III
THROUGHPUT AND ENERGY EFFICIENCY OF THE *μ-engine* AND MULTIPLIER UNIT CONSIDERING LARGE SQUARE MATRICES.

Config.	GOP/s		GOP/sW		Config.	GOP/s		GOP/sW	
	S-i	D-i	S-i	D-i		S-i	D-i	S-i	D-i
a2-w2	15.8	31.0	1405	1349	a6-w6	9.0	13.8	830	528
a3-w2	15.5	26.3	1384	1207	a7-w2	9.3	14.7	920	673
a3-w3	13.2	22.5	1232	998	a7-w3	9.0	15.0	923	671
a4-w2	13.6	24.6	1324	1143	a7-w4	9.2	13.9	791	567
a4-w3	10.0	16.8	838	693	a7-w5	9.0	13.0	727	494
a4-w4	11.5	22.2	894	843	a7-w6	6.8	11.8	418	323
a5-w2	10.3	17.3	897	758	a7-w7	6.8	11.8	395	373
a5-w3	10.5	17.0	784	605	a8-w2	9.3	13.7	948	726
a5-w4	9.0	15.6	939	687	a8-w3	6.8	11.3	637	514
a5-w5	9.0	15.3	836	671	a8-w4	9.3	13.0	818	548
a6-w2	9.8	15.6	831	719	a8-w5	6.4	10.9	429	372
a6-w3	9.1	15.2	897	682	a8-w6	6.6	10.8	415	337
a6-w4	8.4	15.5	781	690	a8-w7	6.8	10.1	392	278
a6-w5	9.0	14.1	799	618	a8-w8	6.7	11.5	366	331
*Mul. 2b	0.46	0.71	84	124	*Mul. 8b	0.46	0.64	92	112

* Baseline BLIS-based GEMM, using the multiplier without the *μ-engine*.

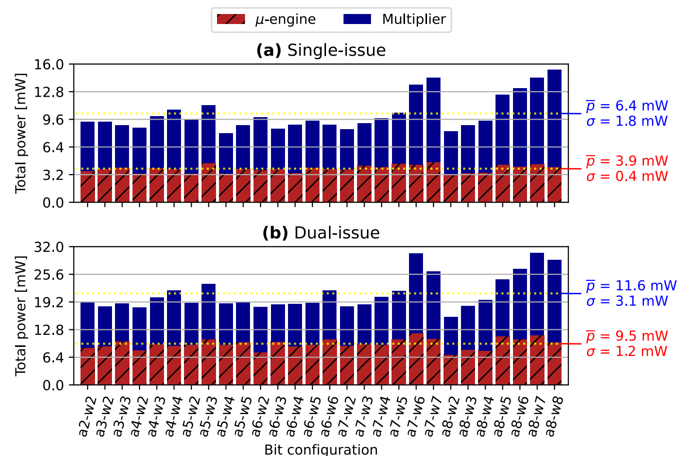


Fig. 11. Average power of the *μ-engine* and the multiplier during computation. $\bar{p}$ and σ denote the mean and standard deviation across bit configurations.

D. Power and Energy Efficiency

We analyze the average power consumption of the *μ-engine* and the multiplier unit during computation, with all the possible bit-width combinations supported by *Mix-GEMM*, in order to accurately estimate the energy efficiency profile of our architecture. The energy efficiency of *Mix-GEMM* is calculated by considering the total power consumption of the *μ-engine* plus the multiplier unit, since the latter is used to calculate the inner product via *binary segmentation*.

Figure 11 shows the power consumption of all bit configurations, and Table III lists the throughput and energy efficiency values. The total power consumption of the single- and dual-issue Sargantana processor is around 129 mW and 178 mW, so the power overhead of the *μ-engine* itself is less than 2.8% and 4.8%, respectively. The most power-hungry element of the *μ-engine* is the *DSU*, accounting for about 60% of the power consumption in both versions, mainly due to the barrel shifters it needs to generate the *input-clusters*.

As shown in Figure 11, the *μ-engine* consumes less power and has a smaller power variation across configurations compared to the multiplier unit. In general, bit configurations with a higher total number of operand bits result in higher power consumption in the multiplier, since more bits are utilized. This contributes to the efficiency boost that can be achieved by using *Mix-GEMM*: lowering the operand bits not only increases throughput but also decreases power consumption.

In terms of energy efficiency, we see from Table III that the single-issue version of *Mix-GEMM* is slightly more efficient than the dual-issue one, with an energy efficiency profile ranging from 1.40 TOP/sW to 366 GOP/sW. As explained in Section IV-A, the dual-issue version has bigger internal scratchpads (four times as large as the single-issue) to optimize performance, which has a penalty in terms of power consumption, especially leakage power. Nevertheless, the single- and dual-issue *μ-engine* versions are able to boost the energy efficiency of the baseline processor multiplier by a factor of $17\times$ and $11\times$, respectively, in the $a2$-$w2$ case.

E. DNN Benchmarks

We benchmark the performance of *Mix-GEMM* for DNN acceleration by analyzing the execution of the dual-issue implementation with 8 state-of-the-art CNNs: AlexNet [13], VGG-16 [14], ResNet-18 [15], Mobilenet-V1 [16], RegNet-x-400mf [17], EfficientNet-B0 [18], ConvNeXt-tiny [19] and ShuffleNet[20]. Similar to Sections IV-A and IV-B, performance is analyzed by emulating the processor in an FPGA. Even though we focus our benchmarks on CNN models, our *Mix-GEMM* architecture can be applied to execute any workload based on matrix-matrix multiplications with quantized data, including Multi-Layer Perceptions (MLPs) and Large Language Models (LLMs) such as GPT-4 [21].

Since the original networks are trained and executed using double-precision floating point arithmetic, we re-train the models using a QAT strategy in order to deploy the networks with mixed-precision narrow integers. We use Brevitas [22], a Pytorch-based library for DNN quantization, to generate quantized versions the DNNs with all the possible combinations of bit-widths supported by *Mix-GEMM* (from 8b to 2b). We retrain the models using ImageNet with four NVIDIA V100 Graphics Processing Units (GPUs).

The first and last layers of the DNNs are kept at 8-bit in order to preserve accuracy, and all other layers are quantized with the target mixed-precision configuration. Even though the *Mix-GEMM* architecture supports arbitrary mixed-precision (different bit-widths on each layer), we configure all hidden layers of our quantized networks in a homogeneous way to simplify the quantization and training process.

Weights are quantized *per-channel* with *scales* computed from the $absmax$ of the weight tensor, while activations are quantized *per-tensor* with *scales* calibrated by analyzing the dynamic range of the different layers. All *scales* and biases are left in floating-point to preserve accuracy. To simplify the training process, both activations and weights are trained with the quantization *zero-point* equal to zero.

All models are retrained using Stochastic Gradient Descent (SGD) with a momentum of 0.9 and weight decay of $1e-4$.

© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2024.3500369

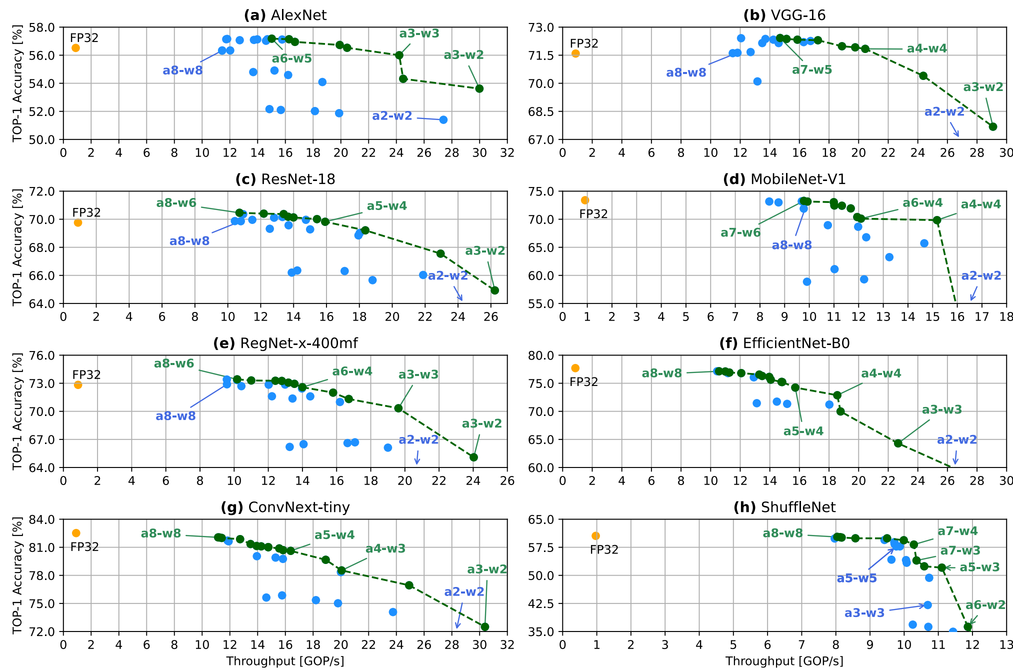


Fig. 12. Performance vs. accuracy of *Mix-GEMM* for the selected CNNs, using all the available bit-widths. The Pareto Frontier is highlighted in green, while suboptimal points are blue. FP32 (orange) is measured exploiting OpenBLAS running on the SiFive U740 processor.

We experiment with different learning rates to search for the highest model accuracy on each configuration. For the ultra-low bit configurations, we adopt a multi-step approach: instead of starting from the double-precision version of the weights, we use an already quantized and retrained version of the model as a starting point. For example, for the $a3-w3$ configuration, we take the $a4-w4$ weights as a starting point; and for $a4-w4$ we start from $a5-w5$. This allows us to achieve higher accuracy in the smallest bit-widths.

We report the average performance of all convolutional and fully-connected layers for each quantized DNN using our dual-issue *Mix-GEMM* implementation, since these layers account for the vast majority of compute time in state-of-the-art DNNs [23]. Figure 12 shows the trade-off plots in terms of TOP-1 classification accuracy and the throughput achieved by our extension. The Pareto-optimal curve is highlighted in the figure, representing the best configurations in terms of performance-accuracy trade-off. Table IV reports the results for the configurations with the highest accuracy, and also with the highest performance given an accuracy tolerance of 5%.

To establish a one-to-one comparison with a commercial RISC-V core, the baseline FP32 performance has been measured by executing the DNNs with the OpenBLAS library [24] on the SiFive U740 RISC-V processor, featuring a 64-bit dual-issue in-order pipeline running at 1.2 GHz.

Using our HW-SW architecture, non-stantard data sizes can be used efficiently to optimize the energy efficiency of DNN execution. For instance, the $a5-w4$ configuration, typically not supported by most accelerators, reaches a 43% performance improvement with respect to the $a8-w8$ configuration on ResNet-18, while retaining almost the same accuracy.

For more aggressive quantizations, the model accuracy starts decreasing substantially, especially in networks such as MobileNet and ShuffleNet, which are highly optimized at the algorithmic level, less redundant, and therefore less resilient to noise. The accuracy of these low-precision configurations could be improved by adopting more sophisticated quantization and training techniques. Arbitrary mixed-precision quantization (i.e. considering different bit-width per layer), which

TABLE IV
ACCURACY AND PERFORMANCE OF THE DNN BENCHMARKS, CONSIDERING (1) THE MOST ACCURATE AND (2) THE FASTEST CONFIGURATION WITH AN ACCURACY DEGRADATION OF $< 5\%$.

Network			TOP-1	GOP/s	GOP/sW
AlexNet		FP32	56.5%	0.89	-
	(1)	a6-w5	57.2%	15.0	659
	(2)	a3-w2	53.6%	30.0	1377
VGG-16		FP32	71.6%	0.89	-
	(1)	a7-w5	72.4%	14.7	561
	(2)	a3-w2	67.7%	29.0	1335
ResNet-18		FP32	69.8%	0.88	-
	(1)	a8-w6	70.5%	10.7	332
	(2)	a3-w2	64.9%	26.2	1204
MobileNet-V1		FP32	73.4%	0.91	-
	(1)	a7-w6	73.3%	9.8	267
	(2)	a4-w4	69.8%	15.2	578
RegNet-x-400mf		FP32	72.8%	0.86	-
	(1)	a8-w6	73.4%	10.2	316
	(2)	a3-w3	70.3%	19.6	870
EfficientNet-B0		FP32	77.7%	0.86	-
	(1)	a8-w8	77.1%	10.6	304
	(2)	a4-w4	72.9%	18.5	706
ConvNext-tiny		FP32	82.5%	0.92	-
	(1)	a8-w8	82.1%	11.1	321
	(2)	a4-w3	78.5%	20.0	826
ShuffleNet		FP32	60.5%	0.97	-
	(1)	a8-w7	60.3%	8.0	219
	(2)	a7-w4	58.2%	10.3	420

© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2024.3500369

TABLE V
STATE-OF-THE-ART: PERFORMANCE AND EFFICIENCY RANGES REPORTED FOR (1) HIGHEST PRECISION AND (2) FASTEST CONFIGURATION.

		Baseline (SiFive U740)	GEMMlowp [25]	DORY [26]	CMix-NN [27]	Brusci et al. [4]	XpulpNN [5]	MPIC [6]	Eyeriss v2 [28]	BitBlade [29]	Marsellus [30]	MixGEMM-v1 [8]	MixGEMM-v2 (This Work)
Architecture		RV64	ARMv8$^{\phi}$	8xRV32†	ARMv7	8xRV32†	8xRV32†	RV32†	Decoupled	Decoupled	RV32+Decoupled	RV64	RV64
Technology		-	-	-	-	-	22	22	65	28	22	22	22
Frequency [GHz]		1.2	1.2	0.26	0.48	0.17	0.4	0.25	0.2	0.195	0.42	1.2	1.2
Area [mm^2]		-	-	-	-	-	0.086‡	0.008‡	2695k gates	0.71	0.46	0.01	0.03 (0.01)$^{\#}$
Precision support		FP32	8b	8b	8b/4b/2b	8b/4b/2b	16b/8b/4b/2b	16b/8b/4b/2b	8b	8b/4b/2b	All 8b-2b	All 8b-2b	All 8b-2b
Mixed-precision?		✗	✗	✗	✓	✓	✗	✓	✗	✓	✓	✓	✓
GOP/s	**(1)**	0.89	5.07	4.2	0.3	6.1	19.8	1.0	154	100$^{\$}$	91	5.3	11.8
	(2)				0.5	2.4	47.9	3.3		1420	569	13.1	29.0
GOP/smm^2	**(1)**	-	-	-	-	-	230	130	-	143	198	530	670$^{\#}$
	(2)						557	429		2029	1236	1310	1580$^{\#}$
GOP/sW	**(1)**	-	-	15.9	1.0	-	700	194	963	2000$^{\$}$	740	11$^{\otimes}$ (222)§	53 (340)§
	(2)				2.0		1100	606		19189	5370	35$^{\otimes}$ (801)§	163 (1335)§
Benchmark		VGG-16	VGG-16	MobileNet-v1	MobileNet-v1	Convolution*	Convolution*	Convolution*	sparse AlexNet	*NA*$^{\$}$	Convolution**	VGG-16	VGG-16

* Input tensor of shape ($H \times W \times F_{in}$) $16 \times 16 \times 32$, and a filter of shape ($F_{out} \times K_{dim} \times K_{dim} \times F_{in}$) $64 \times 3 \times 3 \times 32$
** Input tensor of shape $9 \times 9 \times 64$, and a filter of shape $64 \times 3 \times 3 \times 64$
$^{\$}$ $a8$-$w8$ values extracted from Fig. 17 on [29]. The benchmark network used for the measurements is not specified.
$^{\otimes}$ To establish a one-to-one comparison with [8], we repeated the power estimation with the same physical corners and setup.
† Equipped with custom ISA extension exploiting hardware loops, post-increment load and store, and specialized MAC FUs.
$^{\#}$ Considering the single-issue version.
$^{\phi}$ Includes the NEON SIMD extension.
§ Energy efficiency of the *μ-engine* and 64b multiplier(s).
‡ Overhead of the proposed extension plus the area of the RI5CY dot-product unit(s) [31].

is supported by *Mix-GEMM*, can also be used to improve model accuracy with low precision. For example, [2] quantizes ResNet-18 using a mixed-precision $a4$-$w3$ configuration (i.e. the average bit-width across layers is 4 and 3 bits for activation and weights, respectively), and achieves about 1% higher TOP-1 accuracy than the fixed-precision $a4$-$w3$ we consider in our benchmark, while keeping the same model size.

Our results show that *Mix-GEMM* outperforms the FP32 baseline on all the benchmarked CNNs by a factor of at least $8\times$ in the most conservative bit configurations, up to more than $33\times$ in the most aggressive ones, with many intermediate design points to choose from, as evidenced by Figure 12. This makes our architecture very flexible, since the same *μ-engine* hardware can be used in critical use-cases that require high-accuracy, or in edge-computing applications which can tolerate lower accuracy but need to keep the power as low as possible.

V. Comparison with State-of-the-Art Solutions

Hardware acceleration of quantized DNNs is a widespread research topic [32]. Many works have explored targeting devices such as GPUs [33] and FPGAs [34], but given that our *Mix-GEMM* architecture targets RISC-V CPUs, we will focus mainly on CPU-based architectures in the edge domain. We divide the related literature into three categories: (1) DNN software libraries that exploit existing processors to efficiently compute quantized GEMMs, (2) edge processor architectures that adopt custom ISA extensions with specific FUs, and (3) decoupled DNN accelerators for edge applications, which are typically integrated into a System-on-Chip (SoC) alongside a CPU. Table V summarizes the comparison between our work and the most relevant proposals in the literature.

Optimized Software Libraries. Application-specific libraries targeting commercial edge processors are often used to boost the performance of quantized DNNs, with two examples being Facebook QNNPack [35] and Google GEMMLowp [25]. We compare the performance of *Mix-GEMM* with GEMMLowp by executing our benchmark CNNs on an Arm Cortex-A53, a widely used processor which features a 64-bit, 8-stage, dual-issue in-order pipeline running at 1.2 GHz, and the NEON SIMD extension. As Table V shows, *Mix-GEMM* outperforms GEMMLowp by $2.3\times$ when using the *a8-w8* configuration.

Dory [26] is another library which targets the RISC-V ISA, implementing a framework to deploy DNNs on the GAP-8 processor [36], reaching up to 4.2 GOPS when executing MobileNet-V1 using 8-bit values and 8 cores running in parallel. Comparatively, even when running a single core, *Mix-GEMM* achieves $2.8\times$ better performance with *a8-w8*.

In most of these libraries, the compressed data in main memory must be converted to 8-bit in order to use the SIMD operations offered by current commercial ISAs, which prevents them from using sub-byte operations. These limitations are highlighted in [27], which proposes an inference library for DNNs targeting Arm processors capable of performing 8, 4, and 2-bit mixed-precision computations. Their mixed-precision support allows them to scale their performance up to $2\times$ in energy efficiency and $1.7\times$ in throughput with respect to the 8-bit case. However, their implementation is limited by the lack of mixed-precision and sub-byte SIMD instructions at the ISA level. By extending the RISC-V ISA and addressing these issues, our *Mix-GEMM* architecture is able to outperform [27] by over $30\times$ when executing MobileNet-V1 with *a4-w4*.

Custom ISA Extensions with Specialized Units. Off-the-shelf processors and ISAs have been shown to be inefficient in deploying quantized DNNs with sub-byte data sizes. To address this, several works propose specialized FUs and custom ISA extensions to enable efficient narrow mixed-precision GEMM computations on edge devices.

Bruschi et al. [4] extends the PULP-NN library [3] to support mixed-precision with 8-, 4-, and 2-bit data sizes on an 8-core RISC-V processor. However, their bit-width configurations suffer from performance degradation when lowering the precision from 8-bit to 2-bit, due to the additional *pack* and *extract* instructions that the extension uses. By contrast, *Mix-GEMM* does not suffer from these limitations because the *μ-engine* works directly with compressed data. Using the same convolution benchmark as [4], we scale our performance by $2.7\times$ when comparing 8-bit to 2-bit.

XpulpNN [5] proposes a set of custom RISC-V ISA

© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2024.3500369

instructions and custom FUs to boost the performance of GEMM computations on edge devices. Their microarchitecture features SIMD units supporting from 2x16-bit to 16x2-bit *MAC/cycle*, but does not support mixed-precision computations. Their evaluation targets a 32-bit 8-core edge processor running at 660 MHz, which achieves up to 47.9 GOP/s. In contrast, our solution is able to achieve up to 29.0 GOP/s using a single core. Moreover, [5] requires the dot-product unit of the RI5CY core [37] for its computations, while *Mix-GEMM* only needs the scalar multiplier, already present on any processor's ALU. Considering this, our solution is 2.8× more area efficient than XpulpNN, while also supporting mixed-precision with any bit-width combination from 8 to 2 bits.

MPIC [6] introduces a RISC-V ISA extension to compute mixed-precision dot products with bit-widths from 16- to 2-bit. To avoid an explosion of SIMD instructions, [6] encodes the precision information in a core status register rather than in the instructions. MPIC is integrated in a single-core RISC-V edge processor, which achieves a throughput of up to 3.3 GOP/s (6.5 MAC/cycle) in the $a2$-$w2$ configuration. While their proposed extension adds an area overhead of only 0.002 mm^2, MPIC also requires using the dedicated dot-product unit of the RI5CY core, adding about 0.006 mm^2 [31] to the actual area required by the extension compared to *Mix-GEMM*. Even though MPIC achieves better energy efficiency, the *μ-engine* outperforms it by 8.8× and 3.7× in terms of peak throughput and area efficiency, respectively.

Our work extends and optimizes the previous version of the *Mix-GEMM* [8] architecture, to which we can establish an important comparison. For the sake of clarity, we will refer to [8] as *Mix-GEMM*-v1, and this work as *Mix-GEMM*-v2. In *Mix-GEMM*-v1, we proposed a similar single-issue *μ-engine* architecture to compute inner products with mixed-precision using *binary segmentation*. However, the microarchitecture lacked important features that make *Mix-GEMM*-v2 improve significantly in performance and energy efficiency. On one hand, *Mix-GEMM*-v1 did not have support for hardware loops, so the *μ-engine* needed many more instructions per computation, which hindered its performance and wasted energy in redundant memory accesses. On the other hand, *Mix-GEMM*-v2 adds support for concatenating *μ-vectors* in the *DSU*, which allows the *μ-engine* to be much closer to the theoretical performance upper-bound of *binary segmentation*. These two additions alone increase the throughput and area efficiency of our single-issue *μ-engine* by about 26% and 21%. Another important update is that *Mix-GEMM*-v2 introduces several low-power optimizations to its architecture. We have put a lot of effort into analyzing and optimizing the combinational circuits in the *μ-engine* to avoid unnecessary switching. Regarding the sequential logic, we have introduced clock gating to optimize power consumption during inactive times, such as cache-related stalls; and we have replaced all flip-flop-based memories with latch-based scratchpads. All things considered, the new single-issue *μ-engine* is about 75% more energy efficient. Lastly, we have extended the improved baseline architecture to a dual-issue RISC-V pipeline, which allows us to increase the throughput and energy efficiency up to 2.2× and 1.6×, respectively, with respect to [8].

Decoupled DNN Accelerators. Decoupled accelerators have been extensively studied in the literature and offer outstanding performance, usually at the cost of flexibility. In Eyeriss v2 [28], the authors exploit a bi-dimensional array of 8-bit processing elements and a custom multi-level memory hierarchy, optimized for sparse computations. The accelerator is able to reach 154 GOP/s and 963 GOP/sW, and it requires 2695k NAND-2 gates (which would be equivalent to about 0.54 mm^2 in 22 nm technology).

BitBlade [29] uses a bit-wise summation approach to run mixed-precision DNN workloads with a dataflow architecture. It supports 8-, 4- and 2-bit computation, reaching up to 1.42 TOP/s and 19.2 TOP/sW while requiring 0.71 mm^2.

Marsellus [30] is a heterogeneous 16-core RISC-V SoC which includes the previously discussed XpulpNN extensions, plus a decoupled mixed-precision accelerator called *Reconfigurable Binary Engine* (RBE). The RBE is able to reach up to 637 GOP/s and 12.4 TOP/sW when accelerating DNN layers, with an area overhead of about 0.46 mm^2 (accounting for 19% of their Cluster unit) in 22 nm technology.

Decoupled accelerators dedicate a significant portion of the SoC silicon to achieve high throughput and energy efficiency, as evident from the area results in Table IV. This contrasts with the minimalistic approach of *Mix-GEMM*, which only adds a 2% of area overhead to the dual-issue processor (0.03 mm^2). Compared to Eyeriss, BitBlade and the RBE on [30], our architecture is 18×, 24× and 15× smaller, respectively, making it excel in area-restricted applications. On the other hand, the software stack required by decoupled accelerators is usually very complex, requiring specific offloading mechanisms and coherence management. By contrast, the software libraries proposed in *Mix-GEMM* are relatively simple and easy to use.

With these considerations in mind, while *Mix-GEMM* cannot compete with decoupled accelerators in terms of raw performance, it constitutes a relevant alternative in resource-constrained RISC-V processors in which dedicating a significant portion of the SoC area to accelerator logic is not viable.

VI. Conclusion

We have presented *Mix-GEMM*, an area- and energy-efficient architecture that enables high-performance mixed-precision DNN inference on resource-constrained RISC-V processors. We extend the RISC-V ISA to handle narrow integer GEMM computations with mixed-precision by leveraging the technique of *binary segmentation*, which reshapes the input data to effectively implement SIMD operations in the integer multiplier of the processor.

Our proposed *μ-engine* hardware architecture enables *binary segmentation* with minimal overhead, while adopting different strategies to optimize the performance and energy-efficiency of the computation, such as using scratchpad memories and implementing hardware loops to maximize data reuse. Moreover, we extend the *μ-engine* architecture to a dual-issue implementation that leverages two integer multipliers, effectively doubling the peak throughput while maintaining its small hardware cost. We also create a highly-optimized software library based on BLIS that allows the processor to take full advantage of the *μ-engine* by minimizing cache misses.

© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2024.3500369

We experimentally evaluate *Mix-GEMM* by integrating the extension in an edge RISC-V in-order processor, considering the single- and dual-issue versions. We show that our dual-issue *μ-engine* architecture can reach up to 31.0 GOP/s, improving the performance of the baseline processor by $44\times$ while only adding about 2% of area overhead. By benchmarking *Mix-GEMM* with a set of state-of-the-art CNN workloads, we demonstrate that our solution is more area-efficient than analogous RISC-V extensions, while also being competitive in terms of performance and energy efficiency.

We believe that *Mix-GEMM* represents a step forward in the quest of computing complex DNN workloads with high performance and energy efficiency. Given its high flexibility, support for arbitrary mixed-precision computation and minimal overhead, *Mix-GEMM* presents itself as a relevant alternative to state-of-the-art accelerators, especially in the setting of tightly-constrained edge processors.

REFERENCES

[1] A. Gholami *et al.*, "A Survey of Quantization Methods for Efficient Neural Network Inference," 2021. [Online]. Available: https://arxiv.org/abs/2103.13630

[2] W. Chen, P. Wang, and J. Cheng, "Towards mixed-precision quantization of neural networks via constrained optimization," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 5330–5339.

[3] A. Garofalo *et al.*, "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*, 02 2020.

[4] N. Bruschi *et al.*, "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.

[5] A. Garofalo *et al.*, "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*, vol. 9, no. 3, pp. 1489–1505, 2021.

[6] G. Ottavi *et al.*, "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.

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

[8] E. Reggiani *et al.*, "Mix-GEMM: An efficient HW-SW Architecture for Mixed-Precision Quantized Deep Neural Networks Inference on Edge Devices," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 1085–1098.

[9] V. Pan, "Binary segmentation for matrix and vector operations," *Computers and Mathematics with Applications*, 1993.

[10] F. G. Van Zee and R. A. van de Geijn, "BLIS: A Framework for Rapidly Instantiating BLAS Functionality," *ACM Trans. Math. Softw.*, jun 2015.

[11] V. Soria-Pardos *et al.*, "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.

[12] Micron. LPDDR4 part detail. [Online]. Available: https://www.micron.com/products/memory/dram-components/lpddr4/part-catalog/part-detail/mt53e1g32d2fw-046-it-b

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

[14] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, 2015.

[15] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[16] A. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 04 2017.

[17] J. Xu *et al.*, "RegNet: self-regulated network for image classification," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

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

[19] Z. Liu *et al.*, "A ConvNet for the 2020s," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 11 976–11 986.

[20] X. Zhang *et al.*, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[21] OpenAI *et al.*, "Gpt-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

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

[23] J. Fernández *et al.*, "Towards functional safety compliance of matrix–matrix multiplication for machine learning-based autonomous systems," *Journal of Systems Architecture*, vol. 121, p. 102298, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138376212100206X

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

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

[26] A. Burrello *et al.*, "DORY: Automatic End-to-End Deployment of Real-World DNNs on Low-Cost IoT MCUs," *IEEE Transactions on Computers*, 2021.

[27] A. Capotondi *et al.*, "CMix-NN: Mixed Low-Precision CNN Library for Memory-Constrained Edge Devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2020.

[28] Y.-H. Chen *et al.*, "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, 2019.

[29] S. Ryu *et al.*, "BitBlade: Energy-Efficient Variable Bit-Precision Hardware Accelerator for Quantized Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 6, pp. 1924–1935, 2022.

[30] F. Conti *et al.*, "Marsellus: A Heterogeneous RISC-V AI-IoT End-Node SoC With 2–8 b DNN Acceleration and 30%-Boost Adaptive Body Biasing," *IEEE Journal of Solid-State Circuits*, vol. 59, no. 1, p. 128–142, Jan. 2024. [Online]. Available: http://dx.doi.org/10.1109/JSSC.2023.3318301

[31] A. Garofalo *et al.*, "XpulpNN: Accelerating Quantized Neural Networks on RISC-V Processors Through ISA Extensions," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 186–191.

[32] L. Deng *et al.*, "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey," *Proceedings of the IEEE*, 2020.

[33] G. Li *et al.*, "Unleashing the Low-Precision Computation Potential of Tensor Cores on GPUs," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021.

[34] E. Reggiani *et al.*, "Enhancing the Scalability of Multi-FPGA Stencil Computations via Highly Optimized HDL Components," *ACM Trans. Reconfigurable Technol. Syst.*, no. 3, aug 2021.

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

[36] E. Flamand *et al.*, "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.

[37] M. Gautschi *et al.*, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

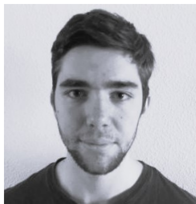
Jordi Fornt received the B.S. degree in industrial electronics from the Universitat Politècnica de Catalunya (UPC) in 2019, and the M.S. degree in electrical engineering and information technology from the Swiss Federal Institute of Technology of Zürich (ETH Zürich) in 2021. During 2020 he worked as a research intern at IBM Research Zürich, with the In-Memory Computing group. In 2021, he joined the Barcelona Supercomputing Center (BSC), where he is pursuing a Ph.D. jointly with UPC, on the topic of energy-efficient AI accelerator design.

© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/TC.2024.3500369

Enrico Reggiani received his M.S. degree in electronic engineering from Politecnico di Milano in 2019, and his Ph.D. in computer architecture at the Barcelona Supercomputing Center (BSC) in 2023. Throughout his doctoral studies, he worked as a research intern at Microsoft Research, Cambridge, and at Huawei Research, Zurich. His research is primarily centered on investigating novel computing systems tailored for high-performance and efficient AI applications.

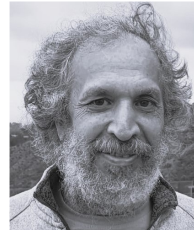
Pau Fontova-Musté received the B.S. degree in industrial electronics and automatic control engineering from the Universitat de Lleida (UdL) in 2018, and the M.S. degree in electronic engineering from the Politècnica de Catalunya (UPC) in 2021. During 2020 he joined the Barcelona Supercomputing Center (BSC) where he currently works as a research engineer.



Narcís Rodas received the B.S. degree in informatics engineering and the M.S. degree in advanced telecommunication technologies from the Universitat Politècnica de Catalunya (UPC) in 2020 and 2024, respectively. Since 2020, he has been working as a research engineer at the Barcelona Supercomputing Centre (BSC) as part of the high-performance domain-specific architectures group. During this time, he has contributed to the development of multiple academic RISC-V cores, some of which have been already taped out. His research interests include vector processing and computer architecture.

Alessandro Pappalardo is a researcher on deep acceleration, with a focus on neural network quantization. He holds a Bachelor's degree in Computer Engineering from Politecnico di Milano, Italy, and a Master's degree in Computer Science from the University of Illinois at Chicago, USA.

Osman Sabri Unsal is a co-manager with the Computer Architecture for Parallel Paradigms Research Group, Barcelona Supercomputing Center, 08034, Barcelona, Spain. He has B.S., M.S. and PhD in Computer Engineering from Istanbul Technical University, Brown University and University of Massachusetts, Amherst, respectively. His research interests include computer architecture, reliability, and low-power computing.



Adrián Cristal Kestelman is leading the architecture development of the vector processor unit in the European Processor Initiative and the Out of Order core at BSC. Since 2006, he has been a co-manager with the Computer Architecture for Parallel Paradigms Research Group. His current research interests include microarchitecture, multicore and heterogeneous architectures, and programming models for multicore architectures. Adrián received a Ph.D. degree in computer science from the Universitat Politecnica de Catalunya, Barcelona.

Josep Altet currently is with the Department of Electronic Engineering, Universitat Politècnica de Catalunya, Barcelona, as Associate Professor. His teaching and research activities are related with analog and digital microelectronic design. He has done research collaborations with research groups of the Université Bordeaux I, Akita University, TIMA (Grenoble), The University of British Columbia, Texas A&M University, CNM-Barcelona and CEA-LETI (Grenoble).

Francesc Moll (Senior Member, IEEE) received his Ph.D. in Electronic Technology from UPC in 1995, and is an associate professor at the Department of Electronic Engineering since 1997. His research career has been focused on reliability and robustness issues relevant to integrated circuit design, especially in advanced technology nodes, such as signal integrity and its impact, manufacturing variability and ultra-low power and voltage circuits. He has design experience in various technology nodes such as 65nm, 45nm, 28nm FDSOI and 22nm FDSOI. He teaches several courses using CAD of IC in the UPC Master in Electronic Engineering. He is an Associate Researcher at the Barcelona Supercomputing Center (BSC) where he leads the team on Synthesis and Physical Design of ICs. He has published over 40 papers in international conferences and journals, serving as reviewer in several international journals and has been invited as keynote speaker at different conferences.

Jaume Abella (PhD, 2005 at UPC, Spain) joined the Barcelona Supercomputing Center (BSC) in 2009 where he co-leads the CAOS group. Formerly, Jaume worked as senior researcher at Intel Corporation (2005-2009). Jaume's research focuses on safety critical systems for the automotive, avionics and space domains among others, including hardware and software safety support for RISC-V architectures, and AI-based autonomous system design, implementation, validation and certification. Other topics of interest are microprocessor reliability and testing, use of commercial HPC MPSoCs in safety-relevant systems, and technology transfer to industry. Jaume is the Principal Investigator of the project Horizon Europe SAFEXPLAIN, and is or has been BSC's PI for several other HE, H2020, Chips JU, KDT JU, ECSEL JU and ARTEMIS JU projects. Jaume has 14 patents and over 250 publications in top peer-reviewed conferences and journals, and has coadvised 30 PhD and master students.

© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.