ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

A mixed-precision RISC-V processor for extreme-edge DNN inference

This is the final peer-reviewed author’s accepted manuscript (postprint) of the following publication:

Published Version:
A mixed-precision RISC-V processor for extreme-edge DNN inference / Ottavi G.; Garofalo A.; Tagliavini G.; Conti F.; Benini L.; Rossi D.. - ELETTRONICO. - 2020-:(2020), pp. 9155071.512-9155071.517. (Intervento presentato al convegno 19th IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2020 tenutosi a Cyprus nel 2020) [10.1109/ISVLSI49217.2020.000-5].

Availability:
This version is available at: https://hdl.handle.net/11585/776845 since: 2021-02-12

Published:

DOI: http://doi.org/10.1109/ISVLSI49217.2020.000-5

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

20 April 2024

This is the final peer-reviewed accepted manuscript of:

G. Ottavi, A. Garofalo, G. Tagliavini, F. Conti, L. Benini and D. Rossi, "A Mixed-Precision RISC-V Processor for Extreme-Edge DNN Inference," 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Limassol, Cyprus, 2020, pp. 512-51.

The final published version is available online at: https://doi.org/10.1109/ISVLSI49217.2020.000-5

Rights / License:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/)

When citing, please refer to the published version.

A Mixed-Precision RISC-V Processor for Extreme-Edge DNN Inference

Gianmarco Ottavi†, Angelo Garofalo†, Giuseppe Tagliavini †, Francesco Conti†*, Luca Benini†* and Davide Rossi†
DEI, University of Bologna, Italy† *IIS lab, ETH Zurich, Switzerland**
{gianmarco.ottavi2, davide.rossi, angelo.garofalo, giuseppe.tagliavini}@unibo.it
{fconti, lbenini}@iis.ee.ethz.ch

Abstract—**Low bit-width Quantized Neural Networks (QNNs) enable deployment of complex machine learning models on constrained devices such as microcontrollers (MCUs) by reducing their memory footprint. Fine-grained asymmetric quantization (i.e., different bit-widths assigned to weights and activations on a tensor-by-tensor basis) is a particularly interesting scheme to maximize accuracy under a tight memory constraint [1]. However, the lack of sub-byte instruction set architecture (ISA) support in SoA microprocessors makes it hard to fully exploit this extreme quantization paradigm in embedded MCUs. Support for sub-byte and asymmetric QNNs would require many precision formats and an exorbitant amount of opcode space. In this work, we attack this problem with status-based SIMD instructions: rather than encoding precision explicitly, each operand's precision is set dynamically in a core status register. We propose a novel RISC-V ISA core *MPIC* (Mixed Precision Inference Core) based on the open-source RI5CY core. Our approach enables full support for mixed-precision QNN inference with 292 different combinations of operands at 16-, 8-, 4- and 2-bit precision, without adding any extra opcode or increasing the complexity of the decode stage. Our results show that MPIC improves both performance and energy efficiency by a factor of 1.1–4.9× when compared to software-based mixed-precision on RI5CY; with respect to commercially available Cortex-M4 and M7 microcontrollers, it delivers 3.6–11.7× better performance and 41–155× higher efficiency.**

Index Terms—**PULP Platform, Embedded-Systems, Deep Neural Networks, Mixed-precision, Microcontroller**

I. INTRODUCTION

Running complex applications on embedded systems like microcontrollers (MCUs) requires optimization on both software and hardware due to severe constraints in terms of memory size, power consumption, and computing power. In an Internet-of-Things (IoT) environment, wireless communication to higher-level nodes often dominates the power budget. Algorithms such as Deep Neural Networks (DNNs), more specifically Convolutional Neural Networks (CNNs) which are state-of-the-art for computer vision and speech recognition, are used in computing at the edge of IoT to reduce the amount of data to transmit by communicating only classes or high-level features instead of the raw sensor data. The complexity of these algorithms typically requires millions of Multiply-Accumulate (MAC) operations and significant memory footprint, where memory is a valuable resource due to its cost in terms of area and power.

An effective way to reduce the memory footprint of DNNs is *quantization*, a technique that reduces inputs and weights to fixed-point formats such as 8- bits, and even sub-byte like 4- and 2- bits [1]–[3]. Banner *et al.* proposed a methodology to quantize both weights and activations to 4-bit with an accuracy drop of only a few percent, not modifying the training and not requiring a full dataset. Rusci *et al.* [1] show how, using mixed-precision quantization for each layer, it is possible to reduce by up to 7× the memory footprint of DNNs, incurring only in a 4% accuracy loss. However, although quantization provides a clear reduction of memory bandwidth visible also in general-purpose processors [4], much of the inference-time benefit is accessible only through customized hardware accelerators [5] or with an FPGA implementation of quantized arithmetic units [6]. To the best of the authors' knowledge, the only recent work taking advantage of quantized formats in software processors is the one presented by Anderson *et al.* [7]. It proposed a software technique exploiting arbitrary bit-precise signed and unsigned integer operations embedding a vector architecture with custom bit-width lanes in fixed-width scalar arithmetic [7]. However, this comes with significant effort in application porting.

From the hardware perspective, the only relevant research work in this field is the reconfigurable Parallel Balanced-Bit-Serial (PBBS) vector processing tile presented by Wu *et al.* [8], which is suitable for improving the efficiency of sub-byte single instruction multiple data (SIMD) computations of heavily leakage-dominated ULP designs. However, code serialization significantly degrades performance and efficiency in near- and super-threshold operating points. On the other hand, the totality of commercial MCUs operates at the finest granularity of 1-byte data [9], [10]. The new ARM [11] ISA specialized for machine learning, implemented by the *Cortex M55* processor, enhances the ARMv8 with extensions similar to the ones presented in [12], such as 8-bit SIMD instructions, loops, and conditional execution extensions. In addition, it provides pipelined execution of load and mac instructions [11] that allows maximizing utilization of MAC units during the execution of regular patterns (e.g., convolutions).

However, similarly to all other commercial cores, the ARM *Cortex M55* does not support natively smaller than 8-bit SIMD instructions. Hence, data have to be presented as a byte for computation, even if it is "packed" in a more compact representation. First, this means that there is no way to exploit the additional parallelism because the datapath is hardwired to 8 bits. Second, in the tight inner loops of the quantized DNN kernels, the cost of unpacking and packing data can be extremely high, leading to up to 2.5× worse performance than directly using 8-bit data, as shown in the results. In our experiments, sub-byte and mixed-precision quantization by itself improved only the implementation feasibility of a network in MCUs (in term of squeezing the network memory footprint),

---

This work was supported in part by OPRECOMP (Open trans-PREcisionCOMPuting) Grant Agreement No. 732631, and WiPLASH (Wireless Plas-ticity for Heterogeneous Massive Computer Architectures) Grant AgreementNo. 863337. Both projects are funded from the European Union's Horizon 2020 research and innovation program.

but not its performance and efficiency in computation [13].

Supporting many different precision formats to avoid data unpacking can be challenging in a general-purpose MCU, because it leads to the proliferation in the number of instructions, saturating the encoding space. Variable-length instructions offer a potential solution to this problem, but only at the cost of code bloat and increased complexity of the decoding stage, which would result in a significant penalty for what concerns the power consumed by the MCU [14]. In this work, we propose a lightweight processor specialization for quantized DNNs leveraging a status-based approach to counter the proliferation of SIMD instructions necessary to support mixed-precision computations. Such instructions do not encode precision explicitly; rather, they encode a "virtual" SIMD instructions, which contain no precision information and the latter is specialized at run-time by setting the precision of the operands in a core status register. In this way, the same virtual instructions can encode a range of operand precision, enabling much higher code efficiency.

The main contributions of this paper are the following: first, we introduce *XMPI*, a RISC-V ISA extension introducing mixed-precision and heavily quantized SIMD instructions to boost the execution of Quantized Neural Network workloads from 16 down to 2 bits; moreover, we extend the functionality of the RI5CY core [12], a state-of-the-art open-source RISC-V core, to support status-based operation. We call this new core *MPIC* (Mixed Precision Inference Core). We then integrated XMPI and added new execution stage functional units to operate at the granularity of 2 and 4 bits.

To validate our design, we deployed the new core into PULPissimo [15], a single-core open-source MCU of the PULP[1] family [16]. We implemented the full layout on a commercially available technology at 22nm FDX from Globalfounderies to evaluate overheads in terms of power, area, and frequency with respect to the baseline RI5CY core. We benchmarked the mixed-precision extended core against RI5CY, ARM Cortex M7, and ARM Cortex M4 cores on a QNN layer with different quantization configurations. The new approach of MPIC avoided the encoding of 200 new instructions keeping power consumption on the level of the baseline RI5CY core. Our results show that the new ISA brings 1.1–4.9× better performance and energy efficiency when compared to software-based mixed-precision on the RI5CY core; moreover, we also compare with commercially available MCUs based on Cortex-M7 and M4 cores, showing that our solution provides a boost of 3.6–11.7× in performance and 41–155× in energy efficiency.

II. BACKGROUND

A. RI5CY Core

RI5CY, used as a baseline for the proposed work, is an open-source core featuring a 4 stage in-order single-issue pipeline based on the RISC-V ISA [10]. It supports the standard RISC-V extensions (I, M, C, and F) but also includes a non-standard extension, called *XpulpV2*, that introduces several features such as hardware loops, bit manipulation instructions, load/store post-increment instructions, SIMD for 16- and 8-bit format (more information can be found at [12]). As later described in Section IV-B, these features provide 4.4× performance in 8-bit kernels when compared to SoA cores such as ARM Cortex M7.

B. Quantized Neural Networks

The QNN layers used for the experimental assessment of our approach adopt layer-wise linear quantization. The quantization process takes care of mapping each tensor to integers values. We have 3 categories of tensors to quantize: input feature maps, output feature maps, and weights ($\mathbf{x}$, $\mathbf{y}$, and $\mathbf{w}$, respectively). A generic real-valued tensor $\mathbf{t}$ that belongs inside the range of $[\alpha_\mathbf{t}, \beta_\mathbf{t})$ can be expressed as:

$$\mathbf{t} = \alpha_\mathbf{t} + \varepsilon_\mathbf{t} \cdot \mathrm{INT}(\mathbf{t}) \tag{1}$$

where $\mathrm{INT}(\mathbf{t})$ is the value of $\mathbf{t}$ mapped to an N-bit integer, $\varepsilon_\mathbf{t} = (\beta_\mathbf{t} - \alpha_\mathbf{t})/2^N$ and $\alpha_\mathbf{t}$ is the bias to shift the value back to its original range. Imposing $\alpha_\mathbf{t} = 0$ for both input and output feature maps (but not weights) gives a QNN that can be trained efficiently by means of linear quantization-aware training [17]. It is possible to work directly on quantized integer values and apply convolution, normalization and activation:

$$\mathrm{INT}(\mathbf{y}) = \mathrm{quant}\Big(\mathrm{conv}\big(\mathrm{INT}(\mathbf{w}), \mathrm{INT}(\mathbf{x})\big)\Big) \tag{2}$$

The result of the convolution $\phi = \mathrm{conv}\big(\mathrm{INT}(\mathbf{w}), \mathrm{INT}(\mathbf{x})\big)$ is still an integer tensor but has to be represented with a larger number of bits than its input (ε_ϕ is smaller than both $\varepsilon_\mathbf{x}$ and $\varepsilon_\mathbf{w}$). We then have the function $\mathrm{quant}(\cdot)$ that first applies batch-normalization (if any) to ϕ and then scales the result to the proper number of output bits.

Mixed-precision QNNs do not impose the same number of bits for activations and weights, opening the possibility to have more sensitive layers and/or tensors to be represented at higher precision while strongly quantizing the rest [1].

C. QNN Execution Model

The execution model for QNNs adopted in this work is based on the PULP-NN library [13], a library composed of QNN kernels optimized to run on PULP systems. These libraries are inspired by the ARM CMSIS-NN [18], but they include additional support for sub-byte quantization of INT-4, -2, -1 integer types. For efficient execution on PULP cores, convolutional layers inference is split into three phases. The *im2col* phase takes the 3D input features and maps it into a 1D vector. The *MatMul* organizes the innermost dot products of the convolution operator as a set of 4×2 matrix multiplications: each inner-loop of the convolution outputs 2 spatially adjacent pixels along 4 consecutive channels. It does so by loading two input buffers and 4 adjacent filters and leveraging the reuse of input elements to ensure a more efficient ratio of MAC/load [13]. Since the results of the matrix multiplication have a higher dimension than their inputs, the last phase (*QntPack*) discretizes the 32-bit outputs of the *MatMul* to their target precision and pack them into 32-bit variables. For this purpose, different discretization techniques are employed depending on the case: 8-bit output uses scaling and clamping [18], while 4- and 2-bit configurations use thresholds [1], [3]. This operation compares the result of the matrix multiplication with a set of thresholds computed at training time, which directly implement the quant function of Eq. 2.

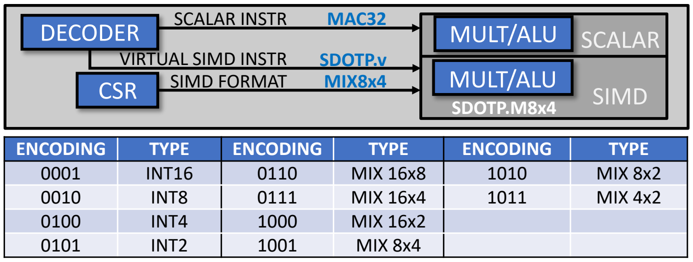
III. ISA EXTENSION

A. Computational Model

The *XpulpV2* extension of the RI5CY core ISA supports 16- and 8-bit SIMD operations. Supporting formats from 16-

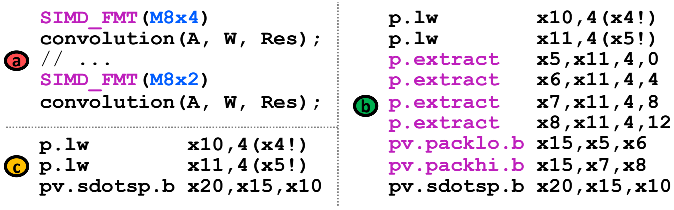
---

[1] https://pulp-platform.org/



ENCODING	TYPE	ENCODING	TYPE	ENCODING	TYPE
0001	INT16	0110	MIX 16x8	1010	MIX 8x2
0010	INT8	0111	MIX 16x4	1011	MIX 4x2
0100	INT4	1000	MIX 16x2		
0101	INT2	1001	MIX 8x4		

Fig. 1. Control signals for SIMD and Scalar instructions. The SIMD instruction is a Sum-of-Dot-Product and the format is a mixed-precision 8x4. The bottom picture contains the encoding of the formats that are contained inside the CSR.



```
SIMD_FMT(M8x4)
convolution(A, W, Res);
// ...
SIMD_FMT(M8x2)
convolution(A, W, Res);
```

```
p.lw          x10,4(x4!)
p.lw          x11,4(x5!)
pv.sdotsp.b   x20,x15,x10
```

```
p.lw          x10,4(x4!)
p.lw          x11,4(x5!)
p.extract     x5,x11,4,0
p.extract     x6,x11,4,4
p.extract     x7,x11,4,8
p.extract     x8,x11,4,12
pv.packlo.b   x15,x5,x6
pv.packhi.b   x15,x7,x8
pv.sdotsp.b   x20,x15,x10
```

Fig. 2. **a**) MPIC Functions Call changing precision before executing operations; **b**) RI5CY inner loop with data packing/unpacking overhead; **c**) MPIC inner loop with MAC instructions executed directly.

Instruction	**Description**
ALU SIMD Instr.	
pv.add[.sc(i)] pv.sub[.sc(i)] pv.avg(u)[.sc(i)]	rD[i] = rs1[i] + rs2[i] rD[i] = rs1[i] - rs2[i] rD[i] = (rs1[i] + rs2[i]) » 1
Vector Comparison Instr.	
pv.max(u)[.sc(i)] pv.min(u)[.sc(i)]	rD[i] = rs1[i] > rs2[i] ? rs1[i] : rs2[i] rD[i] = rs1[i] < rs2[i] ? rs1[i] : rs2[i]
Vector Shift Instr.	
pv.srl[.sc(i)] pv.sra[.sc(i)] pv.sll[.sc(i)]	rD[i] = rs1[i] » rs2[i] Shift is logical rD[i] = rs1[i] » rs2[i] Shift is arithmetic rD[i] = rs1[i] « rs2[i]
Vector ABS Instr.	
pv.abs	rD[i] = rs1[i] < 0 ? -rs1[i] : rs1[i]
Dot Product Instr.	
pv.dotup[.sc(i)] pv.dotusp[.sc(i)] pv.dotsp[.sc(i)] pv.sdotup[.sc(i)] pv.sdotusp[.sc(i)] pv.sdotsp[.sc(i)]	rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7] rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7] rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7] rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7] + rs3 rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7] + rs3 rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7] + rs3

TABLE I
LIST OF INSTRUCTIONS EXTENDED BY XMPI

down to 2-bit, and all the permutations of mixed-precision operations, would require 10 different encodings per each supported MAC-SIMD instruction for a total of 292 instructions versus 92 of the baseline core. This would require 4 bits, while only one is available for this purpose in the current RISC-V encoding [2]. In the MPIC core, we eliminate the problem of encoding space by using virtual instructions. As depicted in Figure 1, for regular scalar instructions the decoder directly produces control signals towards the *ex_stage* (e.g., in the case of a MAC with 32-bit operands); virtual SIMD instructions require additional information from the status registers (CSR) to be specialized (e.g., in the case of an 8-bit by 4-bit sum-of-product).

Application code requires explicit modifications to use SIMD virtual instructions. In Figure 2 **a)**, we have an example of a QNN with multiple layers using different precisions. Here we can note how before the function calls, we set the precision with the *SIMD_FMT* macro, which writes the appropriate format encodings into the CSR. If we "zoom" inside the functions and get to the inner loop of the *8x4* mixed-precision kernel, we can see how supporting directly in hardware this new format benefits the computation. In Figure 2 **b)**, we show the normal instruction flow using RI5CY: first, we load the activations and weights from memory; then, we unpack four of the eight operands in the 32-bit register containing the current weight. Once unpacked, they have to be packed again into 8-bit operands, to take advantage of RI5CY's 8-bit vector MAC instruction. On the other hand, MPIC only requires to load and execute the vectorial MAC (Figure 2 **c)**). Thus, it saves two-thirds of the instructions on the inner loop when running on data smaller than a byte (or mixed-precision).

In Figure 3, we illustrate how a matrix multiplication kernel works in the case of mixed-precision operands. Having the same instructions that deal with both mixed-precision and uniform-precision operations requires added logic for the management of the input with smaller operands; by construction, we always map it to input B. As shown in Figure 3, input B can remain stationary for multiple MAC instructions before new data is needed to be fetched. In the 8x2-bit example of the figure, choosing the correct group of 4 operands out of the 16 is crucial for the correctness of the result. To this end, we designed a controller to deal with this problem, which is explained in more detail in Section III-C.

B. Virtual Instructions

Table I lists the instructions in the XMPI extension. The instructions are derived from a subset of *XpulpV2*, where they are available in 16-, and 8-bit precision only. We extended them with additional support for symmetric 4- and 2-bit precision and, for the dot-product instructions, with also mixed-precision support. We have different versions of these instructions: the *sc* variant is an operation between a scalar and a vector; the *i* variant uses the value from the immediate field instead of a register; finally, we support signed and unsigned variants (dotp instructions also have a hybrid unsigned-signed).

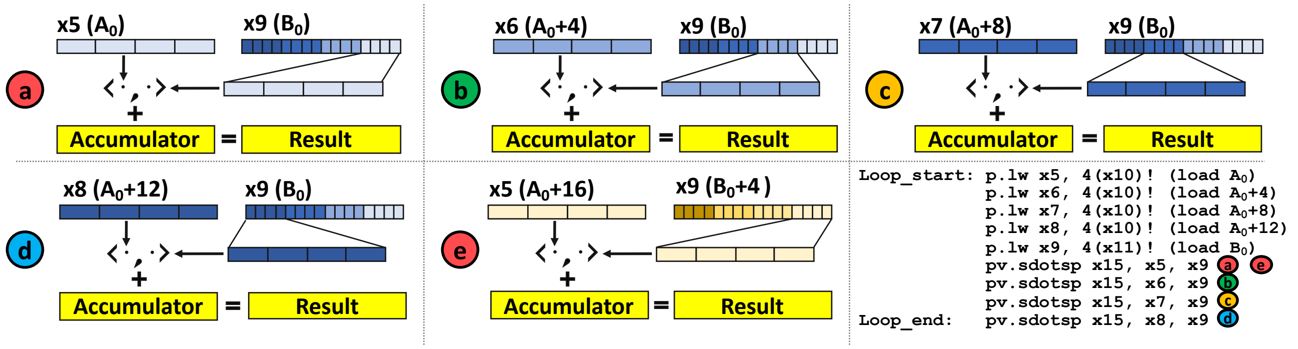
C. Microarchitecture

Figure 4 shows the diagram of the MPIC pipeline, highlighting the IPs modified to implement the new extension. The logic required to decode the format for SIMD instructions has been removed by the decoder and moved to the CSR, which now has a register dedicated to this task (the orange signal feeds the precision to the functional units). Moreover, two additional registers have been added for managing mixed-precision operations.

Dot-product Unit: The baseline RI5CY core already supported the cumulative dot product operation for 16- and 8-bit MAC operations; it consists of two distinct sets of multipliers, one for each data size [12]. The intermediate multiplications are fed to an adder tree to sum all the contributions. It accepts two 16-bit or four 8-bit operands packed in one 32 bit register, and an optional third input register used as an accumulation register. To extend its support to 4 and 2 bits, we followed the same principle, adding another set of multipliers and an adder tree for each supported format. This configuration enables the execution of 8 and 16 operations per cycle for 4- and 2-bit, respectively, paying the cost for more area but with no impact on the critical path of the design. On the other hand, we apply a power management policy for the unused SIMD units by

---

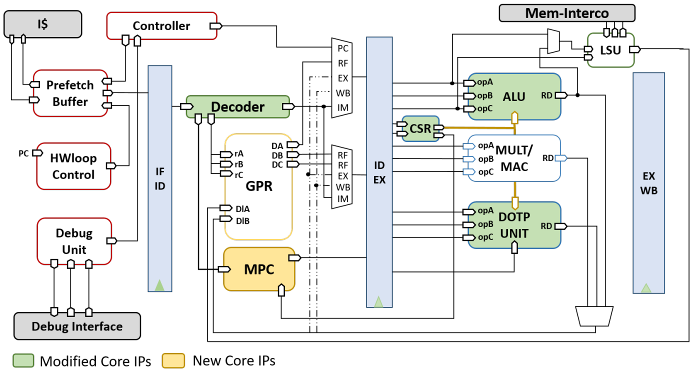
[2] https://www.pulp-platform.org/docs/ri5cy_user_manual.pdf



a
x5 (A_0)
x9 (B_0)
Accumulator
=
Result
b
x6 (A_0+4)
x9 (B_0)
Accumulator
=
Result
c
x7 (A_0+8)
x9 (B_0)
Accumulator
=
Result
d
x8 (A_0+12)
x9 (B_0)
Accumulator
=
Result
e
x5 (A_0+16)
x9 (B_0+4)
Accumulator
=
Result

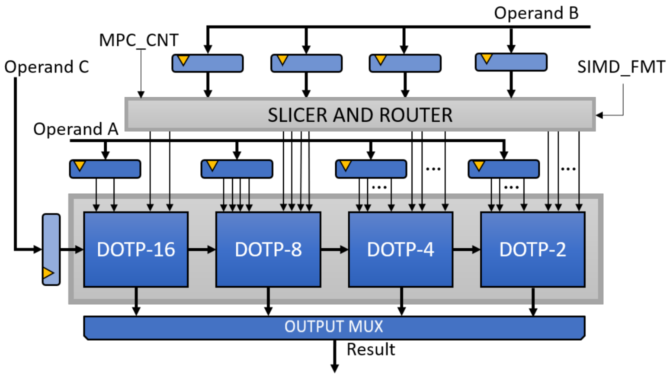
```
Loop_start: p.lw x5, 4(x10)!  (load A0)
            p.lw x6, 4(x10)!  (load A0+4)
            p.lw x7, 4(x10)!  (load A0+8)
            p.lw x8, 4(x10)!  (load A0+12)
            p.lw x9, 4(x11)!  (load B0)
            pv.sdotsp x15, x5, x9 a e
            pv.sdotsp x15, x6, x9 b
            pv.sdotsp x15, x7, x9 c
Loop_end:   pv.sdotsp x15, x8, x9 d
```

Fig. 3. Matrix multiplication between operands of size 8- and 2-bit. Vector B contains four times the operands of Vector A requiring the fetch of 3 more vectors to "exhaust" Vector B. In each step, we have a group of operands unpacked (via Hardware) from Vector B, extended to match the size of Vector A, and finally execute the dot-product between the vectors where the partial result is added to the accumulator to get the final result. On the bottom right, we can see the kernel assembly: `p.lw` are post increment loads that increment by 4 the pointer after load, and `pv.sdotsp` is a signed sum of dot-product.



I$
Controller
Mem-Interco
LSU
Prefetch Buffer
HWloop Control
PC
Debug Unit
Debug Interface
IF ID
Decoder
GPR
rA
rB
rC
DA
DB
DC
DIA
DIB
MPC
CSR
PC
RF
EX
WB
IM
ID EX
opA
opB
opC
ALU
MULT/ MAC
DOTP UNIT
RD
EX WB
Modified Core IPs
New Core IPs

Fig. 4. Pipeline of the MPIC core.



Operand B
MPC_CNT
Operand C
SIMD_FMT
SLICER AND ROUTER
Operand A
DOTP-16
DOTP-8
DOTP-4
DOTP-2
OUTPUT MUX
Result

Fig. 5. Extended dot-product block.

means of clock gating of the input registers of the units not involved in the current computation, as shown in Figure 5.

For what concerns mixed-precision operation, we have operands with different size multiplied together; this implies that one of the input registers contains a higher number of operands than the other, and when we execute a dot-product, one of the input registers is fully utilized, while only a part of the second one is needed. This requires two actions: first, we need to select a sub-group of the second input register (we call it *input B*); second, that sub-group has to be fed to the correct size multiplier; e.g., if we consider an *8x4* MAC, the correct sub-group of operands from input B has to be routed to the 8-bit multiplier. In Figure 5, we depict the whole dot-product module. The *slicer and router* block is used to select and direct the correct sub-group of operands to the various *dotp* multipliers; it also sign-extends the smaller operands to match the size of the larger operands. This block is controlled by two signals: *MPC_CNT* is used to select the sub-groups of operands (discussed later), and *SIMD_FMT* specifies which type of operands to select (taken directly from the CSR).

A mechanism to correctly choose the sub-group of inputs for register B is needed, so we designed a small controller dedicated to this task.

Mixed-Precision Controller: To behave as shown in Fig. 3, the mixed-precision controller (MPC) contains a counter that is used to select which subgroup of operands to use. The counter is increased only if the following requisites are met: the *ID_STAGE* is decoding, which avoids the counter to go up in case of stalls; the instruction is a MAC; the format set by the CSR is mixed-precision. The counter resets by itself depending on the current format used (for example, 8x4 counts up to 2, while 8x2 counts up to 4). However, to implement the execution model explained in Sec. II-C, a single counter is not enough due to data reuse. This causes each sub-group of operands to be used multiple times before switching to the next one. To work around this problem, we added a second counter that can be programmed to count the number of MACs to execute before changing the sub-group of operands (e.g., in the kernel 4x2 we execute 8 MAC before switching to the next sub-group). The value of the sub-group is also written inside the CSR; this can be changed by writing directly to it, making it possible to choose the group of operands manually.

D. Compiler support

We integrated the support to the mixed-precision instruction semantics into the PULP GNU toolchain[3]. The GCC front-end does not require any modification since programmers use a single integral type (i.e., `int`) and possibly modify the precision of the operations setting the status register; this approach is suitable for both homogeneous and mixed-precision operations. At the middle-end level, we disabled automatic loop unrolling for the loops that include mixed-precision instructions, intending to avoid inconsistencies with the internal counter of the mixed-precision controller. For

---

[3] https://github.com/pulp-platform/pulp-riscv-gnu-toolchain

Power consumption results [mW] @250 MHz			
	RI5CY	**MPIC**	Overhead
GP App	5.39	5.30	-2%
8-bit MatMul	5.36	5.39	1%
4-bit MatMul	-	5.46	2%
2-bit MatMul	-	5.38	0%

Area Results [μm^2]			
	RI5CY	**MPIC**	Overhead
SoC	1002681	1004273	0.2%
core	15755	17584	11%
ex_stage	6592	7655	16%
id_stage	5276	5673	7.5%

TABLE II
IMPLEMENTATION RESULTS IN AREA AND POWER

the same reason, we inhibited the reordering of the mixed-precision instructions into the compiler backend.

IV. RESULTS

The evaluation of the MPIC has been performed on two fronts. The first one is the physical implementation, where we extracted values of power, area, and frequency, which were used to compare our approach to the baseline core. The second front was the performance assessment with benchmarks, where we executed a QNN layer from 8 bit down to 2 using uniform and the various mixed-precision variants, also providing a comparison with a commercially available MCUs sporting Cortex M7 and Cortex M4 cores.

A. *Implementation Results*

The experimental results presented in this section are based on both RI5CY and MPIC cores integrated into the PULPissimo SoC, which features a full set of peripherals, a DMA subsystem, and 512 kB of SRAM memory. We synthesized the SoC with Synopsys Design Compiler-2016.03 using Global Foundries 22 nm FDX technology, place & route was performed with Cadence Innovus-15.20.100 in the worst-case corner (SSG 0.59v, -40°C/125°C); power analysis was done at 250 MHz with Typical Corner 0.65V at 25°C using Synopsis PrimeTime. We performed different runs, one to test the max frequency of the SoC, while the other we set the constraint at 250 MHz aiming at maximizing energy efficiency for power analysis purposes.

From the results of the max-frequency synthesis run, we observed a negligible reduction in the maximum operating frequency from 511 to 505 MHz (1% slowdown). For what concerns power analysis, four different scenarios have been profiled: one to evaluate the impact of the introduced extensions on general-purpose code, while the other 3 while using the modified dot-product unit while executing 8-, 4- and 2-bit QNN kernels.

The results are reported under the power consumption section in Tab. II. Surprisingly, MPIC consumes slightly less power in general-purpose applications, thanks to the addition of clock-gating for unused dotp modules (Sec. III-C), which was shared among all the dotp units in RI5CY. Overall, the table shows that power results are all 2% within each other, well inside the margin of error, telling us that the changes made did not significantly impact the overall efficiency of the core.

The second section of Table II reports area results. The core has an 11% overhead given by the extension of the core for status-based operation. The ex_stage has a 16% overhead for the added logic to support the new precision operations, while the id_stage is larger by 7.5%; this effect is due to several factors: the main contribution is due to the registers that have been added for operand isolation (the id_stage contains the ID/EX pipeline stage) for MAC operations, while a secondary contribution is that of the mixed-precision controller. Overall, the SoC overhead is around 0.2% since the 512 kB of SRAM occupies the most area.

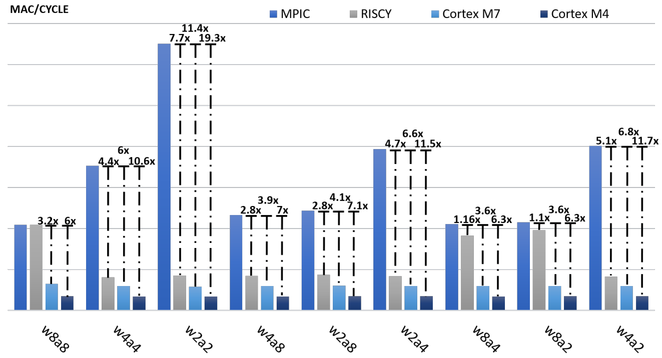


Fig. 6. Performance expressed in MAC/Cycle.

B. *Benchmarking*

To show the performance benefits of supporting these new precision formats, we choose a QNN layer with different configurations for input/output and weights. The input tensor size is 16x16x32, while the filter is 64x3x3x32; this configuration is among the ones featuring the best performance on the targeted architectures. The devices used for this comparison are the baseline core RI5CY, MPIC, STM32H7 equipped with a Cortex M7 (40 nm technology) [19], and STM32L4 with Cortex M4 (90 nm); results consider the STM32H7 running at 480MHz and STM32L4 at 80MHz [20].

Figure 6 shows the results in terms of MACs per Cycle. The different configurations in the chart are denoted in the X-axis by the size of the activations and weights. Analyzing the charts, we can see different trends. The Cortex M7 and M4 have lower performance compared to the MPIC in all the configurations, or even in comparison with RI5CY core (in the case of 8-bit uniform quantization or 8-bit weights). This is due to the high overhead introduced by the unpacking of data before the execution of MAC, but also to the fact that only up to two 16-bit MACs in one cycle are supported for both ARM Cortex.

The RI5CY core presents about 2.1 MAC/Cycle for the first case, well above the Cortex M7/M4 and on par with MPIC, because both support 8-bit MACs. When going to sub-byte configurations, it suffers the same fate as the ARM (except for 8-bit weights) core due to the additional overhead introduced by unpacking data in the *MatMul* phase (Sec. II-C). We can see that compared to the Cortex M7 or M4, the better performance of RI5CY is due to more efficient ISA: load/store post-increment, hardware loops, and the possibility to execute 4 MACs per cycle at 8-bit precision. In contrast, MPIC does not require to unpack data before execution; data can be fed directly to the dot-product unit, resulting in a peak of 6.5 MACs per cycle in the 2-bit uniform layer. When looking at 8-bit weights, we can see that the performance is close to the 8-bit uniform quantization; this is because unpacking is done in the *im2col* execution phase, which is way less computationally intensive and does not impact execution as much as the inner-loop of the kernel [13].

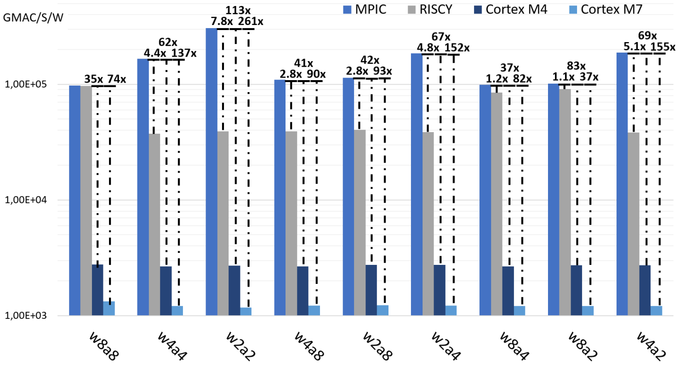


Fig. 7. Energy efficiency expressed in GMAC/s/W.

Significantly, mixed-precision QNN kernels also do not suffer any performance hit, thanks to unpacking done in hardware. The performance of 8x4 and 8x2 kernels are close to the 8-bit uniform kernel, likewise for the 4-bit one. This is because the selection of the dotp module (Fig. 5) is tied to the size of the greater operand (e.g., 8x4 uses 8-bit multipliers). However, we can see that the performance is slightly better than their equivalent uniform case, thanks to the higher operational intensity. If we perform a mixed-precision 8x4 operation, operand b needs to fetch fewer data from memory, since its register can hold twice as many operands as the register containing a. Another factor that impacts mixed-precision operation is the quantization process (*QntPack*). Focusing on the chart for activations of 4- and 2-bit, the performance is marginally worse than when we have 8-bit activations.

In contrast with performance in MAC/cycle, energy efficiency (expressed in GMAC/s/W) takes into account also physical design parameters such as the fabrication technology and the operating voltage and frequency. For the Cortex M7 and M4, we used an implementation from ST-Microelectronics consuming ∼234 mW at 480 MHz [19] and 10 mW at 80 MHz [20], respectively; while we used the power consumption figures reported in Table II for the RISC-V SoCs. In Figure 7, we can see that the lower performance of the Cortex M7 is emphasized even more by the technology factor, having a peak of 1.27 GMAC/s/W and being from 74x to 255x less efficient in these workloads compared to MPIC. The Cortex M4 is way more efficient than the Cortex M7 but still falls short when compared to RISC-V cores, being from 35x to 113x less efficient. For the RI5CY core, we have a slight disadvantage of 1% only in the 8-bit case, while in all other scenarios, the results are qualitatively similar to the performance ones.

V. CONCLUSION

In this work, we presented an alternate way to deal with a saturated encoding space. We extended the ISA to support sub-byte and mixed-precision formats aiming at improving the performance of QNN via removing the overhead caused by unpacking data before computation. The MPIC-based SoC implementation resulted in an area overhead of 11% when compared to the baseline core while having a negligible impact on frequency and power and so not compromising the general-purpose nature of the RI5CY core. The performance gain ranges from 1.1× to 7.7× when compared to the baseline during the execution of a QNN layer, and from 3.6× up to 19.3× in regard to the Cortex M7 and M4. The energy efficiency peaks at 303 GMAC/s/W for the 2-bit convolution and ranges from one to two orders of magnitude higher when compared with ARM counterpart, providing a solution that is considerably more efficient than commercially available MCUs solutions for QNN inference.

REFERENCES

[1] M. Rusci, A. Capotondi, and L. Benini, "Memory-driven mixed low precision quantization for enabling deep network inference on micro-controllers," *arXiv preprint arXiv:1905.13082*, 2019.

[2] B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst, "Minimum energy quantized neural networks," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2017, pp. 1921–1925.

[3] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[4] A. Stojanov, T. M. Smith, D. Alistarh, and M. Püschel, "Fast quantized arithmetic on x86: Trading compute for data movement," in *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*, Oct 2018, pp. 349–354.

[5] B. Moons and M. Verhelst, "An energy-efficient precision-scalable convnet processor in 40-nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 903–914, April 2017.

[6] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, and et al., "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 26–35. [Online]. Available: https://doi.org/10.1145/2847263.2847265

[7] A. Anderson, M. Doyle, and D. Gregg, "Scalar arithmetic multiple data: Customizable precision for deep neural networks," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, June 2019, pp. 61–68.

[8] B. Wu and I. Wey, "Parallel balanced-bit-serial design technique for ultra-low-voltage circuits with energy saving and area efficiency enhancement," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 1, pp. 141–153, Jan 2018.

[9] ARM, "Arm architecture reference manual armv8," 2013-2020, https://developer.arm.com/docs/ddi0487/latest/arm-architecture-reference-manual-armv8-for-armv8-a-architecture-profile.

[10] U. o. C. B. Andrew Waterman; Krste Asanovi; SiFive Inc., CS Division; EECS Department, "The risc-v instruction set manual, volume i: User-level isa," April 2019.

[11] D. E. Joseph Yiu, "Introduction to the arm cortex-m55 processor. available online: https://pages.arm.com/cortex-m55-introduction.html," February 2020.

[12] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "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, Oct 2017.

[13] 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*, vol. 378, no. 2164, p. 20190155, 2020.

[14] O. Azizi, A. Mahesri, B. C. Lee, S. J. Patel, and M. Horowitz, "Energy-performance tradeoffs in processor architecture and circuit design: a marginal cost analysis," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 26–36, 2010.

[15] P. D. Schiavone, D. Rossi, A. Pullini, A. Di Mauro, F. Conti, and L. Benini, "Quentin: an ultra-low-power pulpissimo soc in 22nm fdx," in *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*. IEEE, 2018, pp. 1–3.

[16] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, "Pulp: A parallel ultra low power platform for next generation iot applications," in *2015 IEEE Hot Chips 27 Symposium (HCS)*, Aug 2015, pp. 1–39.

[17] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "Pact: Parameterized clipping activation for quantized neural networks," *arXiv preprint arXiv:1805.06085*, 2018.

[18] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," *arXiv preprint arXiv:1801.06601*, 2018.

[19] STMicroelctronics, "Stm32h743 datasheet," 2018, https://www.st.com/resource/en/datasheet/stm32h743bi.pdf.

[20] STMicroelctronics, "Stm32l476 datasheet," 2018, https://www.st.com/resource/en/datasheet/stm32l476je.pdf.