

ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

XpulpNN: Enabling Energy Efficient and Flexible Inference of Quantized Neural Networks on RISC-V Based IoT End Nodes

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

Published Version:
Garofalo A., Tagliavini G., Conti F., Benini L., Rossi D. (2021). 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, 9(3), 1489-1505 [10.1109/TETC.2021.3072337].

Availability:
This version is available at: https://hdl.handle.net/11585/847003 since: 2022-01-23

Published:

DOI: http://doi.org/10.1109/TETC.2021.3072337

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)

05 March 2026

This is the final peer-reviewed accepted manuscript of:

Garofalo A., Tagliavini G., Conti F., Benini L., Rossi D., "XpulpNN: Enabling Energy Efficient and Flexible Inference of Quantized Neural Networks on RISC-V Based IoT End Nodes," in IEEE Transactions on Emerging Topics in Computing, vol. 9, no. 3, pp. 1489-1505, 1 July-Sept. 2021, doi: 10.1109/TETC.2021.3072337.

The final published version is available online at:

https://ieeexplore.ieee.org/document/9406333

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.

XpulpNN: Enabling Energy Efficient and Flexible Inference of Quantized Neural Networks on RISC-V based IoT End Nodes

Angelo Garofalo*, Giuseppe Tagliavini *, Francesco Conti *, Luca Benini * †, and Davide Rossi *
*Department of Electrical, Electronic and Information Engineering (DEI), University of Bologna, Italy
†IIS Integrated Systems Laboratory, ETH Zurich, Switzerland

***Abstract*—Heavily quantized fixed-point arithmetic is becoming a common approach to deploy Convolutional Neural Networks (CNNs) on limited-memory low-power IoT end-nodes. However, this trend is narrowed by the lack of support for low-bitwidth in the arithmetic units of state-of-the-art embedded Microcontrollers (MCUs). This work proposes a multi-precision arithmetic unit fully integrated into a RISC-V processor at the micro-architectural and ISA level to boost the efficiency of heavily Quantized Neural Network (QNN) inference on microcontroller-class cores. By extending the ISA with nibble (4-bit) and crumb (2-bit) SIMD instructions, we show near-linear speedup with respect to higher precision integer computation on the key kernels for QNN computation. Also, we propose a custom execution paradigm for SIMD sum-of-dot-product operations, which consists of fusing a dot product with a load operation, with an up to 1.64 × peak MAC/cycle improvement compared to a standard execution scenario. To further push the efficiency, we integrate the RISC-V extended core in a parallel cluster of 8 processors, with near-linear improvement with respect to a single core architecture. To evaluate the proposed extensions, we fully implement the cluster of processors in GF22FDX technology. QNN convolution kernels on a parallel cluster implementing the proposed extension run 6 × and 8 × faster when considering 4- and 2-bit data operands, respectively, compared to a baseline processing cluster only supporting 8-bit SIMD instructions. With a peak of 2.22 TOPs/s/W, the proposed solution achieves efficiency levels comparable with dedicated DNN inference accelerators and up to three orders of magnitude better than state-of-the-art ARM Cortex-M based microcontroller systems such as the low-end STM32L4 MCU and the high-end STM32H7 MCU.**

***Index Terms*—IoT, Quantized Neural Networks, Embedded Systems, Fixed-Point Arithmetic, Low-bitwidth Integer Arithmetic, Low-power design, RISC-V, Parallel Ultra Low-Power Computing.**

I. Introduction

In the last years, we are assisting at an exponential growth of the Internet-of-Things (IoT) interconnected devices pervading several application domains such as agriculture, health monitoring [1], structural health monitoring [2]. This scenario requires the IoT end-nodes to acquire data from low-power sensors and send it wirelessly through the network[3], after applying signal processing algorithms.

Machine Learning (ML) algorithms, including state-of-the-art Deep Learning (DL), not only empower the IoT nodes with smart capabilities widening the IoT applications with DL-enhanced tasks but they provide "information distillation" solutions to extrapolate actionable information from the raw data acquired by sensors. Their capability of "squeezing" raw data in a much more semantically dense format (e.g., extracting classes, high-level features, symbols) allows the wireless transmission of a limited amount of condensed information. This feature alleviates the traffic on the IoT network and reduces security and reliability issues, nowadays exacerbated by the significant increase of raw data flowing through the network [4]. The clear benefits of embedding intelligence on IoT nodes have attracted the attention of a wide research area, intending to deploy DL functionality at the extreme-edge of the IoT. This effort has to run against the high computational and memory requirements of leading DL methods that clash with the usual scarcity of computing and memory resources of deeply embedded systems powered by batteries or energy harvesters.

One of the key characteristics of Deep Neural Networks (DNN) exploited in all different flavors of computing platforms from high-performance to low-power is their resiliency to strong arithmetic quantization. While DNN tasks typically run on GPUs or FPGAs devices in data centers, characterized by a power envelope that is orders of magnitude higher than what is sustainable on extreme-edge devices, dedicated DNN accelerators are starting to gain traction for ultra-low power devices [5], [6]. However, these heavily specialized solutions are often not affordable in the extremely cost-conscious and fragmented IoT market. MCUs are the standard computing platforms, thanks to their flexible software programmability, low-cost and low-power characteristics. However, they present severe limitations in memory footprint and computing resources, preventing them from meeting latency and accuracy requirements of advanced DL-enhanced applications. This is the field where arithmetic quantization can provide a breakthrough enabling the execution of state-of-the-art DNN at the edge of the IoT.

More specifically, to reduce the model size of modern DNN topologies and make them fit the limited storage available on MCUs, recent progress in DL training methodologies has introduced novel quantization methods. These techniques represent the network weights and activations with 8-bits (or even smaller) data types, usually adopting fixed-point formats, incurring a limited or negligible loss in accuracy[7], [8], [9]. The limited footprint and the good accuracy achieved

Manuscript received December 1, 2012; revised August 26, 2015. Corresponding author: A. Garofalo (email: angelo.garofalo@unibo.it).

make Quantized Neural Networks (QNNs) the natural target to embed intelligence on MCU-based platforms and encourage many efforts by industry and academia to enable the QNN computation on microcontroller-based systems. In this context, it is worth citing the CMSIS-NN library [10] developed by ARM for 16-bit and 8-bit QNNs on Cortex-M based systems and PULP-NN [11], an open-source library [1] targeting RISC-V processors and supporting heavily quantized NNs on 8-, 4-, 2-, 1-bits data, as well as Mixed-Precision QNNs. On the hardware side, modern MCUs lack support at the Instruction Set Architecture (ISA) level for low bit-width integer Single-Instruction-Multiple-Data (SIMD) arithmetic instructions. Modern MCUs adopting commercial ISAs only support 16-bits (e.g., ARMv7E-M) or 8-bits (e.g., RV32IMCXpulpV2 [12], ARMv8.1-M [13]) data. Hence, sub-byte quantization remains an effective technique to compress the footprint of DNN models on top of these devices [7], but it incurs in performance and energy overhead during the computation, as demonstrated in [11], [14]: low precision data has to be unpacked to the lowest precision operand supported by the underlying hardware and then packed into SIMD registers before feeding the multiply-accumulate (MAC) units [11].

In this work, we tackle this problem by designing an energy-efficient multi-precision arithmetic unit, targeting the computing requirements of low bit-width QNNs, with the support for sub-byte SIMD operations (8-, 4-, 2-bits). To provide the highest flexibility, the unit is integrated into a cluster of MCU-class RISC-V cores, provided with a new set of ISA domain-specific instructions, namely *XpulpNN*.

The main contributions of this paper are the following:

- We design a multiple-precision Dot-Product (*Dotp*) Unit featuring single-cycle latency operations on SIMD vectors of 16- down to 2-bit precision elements. We present micro-architectural optimizations and power-aware techniques to achieve high energy proportionality and efficiency.
- We integrate the unit into an open-source RISC-V processor [12], further extending the core with novel fused mac&load instructions, aiming at increasing the utilization of the SIMD Dot-Product unit in the core towards the theoretical bound of 1 (0.92 in the best case scenario).
- To exploit the low bit-width integer SIMD computation enabled by the designed hardware, we extend the ISA of the core with domain-specific instructions, namely *XpulpNN*. We map the *XpulpNN* extensions on top of the extended core, and we enhance the GCC toolchain with machine descriptions of the new instructions to have a full hardware-software interface;
- We integrate the extended core in an eight cores Parallel Ultra-Low-Power (PULP) computing cluster, showing that we improve the performance of QNN kernels almost linearly with respect to the single-core execution;
- We implement the PULP cluster integrating the proposed core in GF 22nm FDX technology to evaluate the area, power, and performance overhead of the core and the whole system with respect to the baseline RI5CY core and the PULP cluster integrating it, respectively;
- We compare a PULP system with the proposed extension with state-of-the-art architectures and software. When running QNN convolution layers, we are able to demonstrate at least two orders of magnitude better performance and energy efficiency with respect to commercially available solutions such as STM32H7 and STM32L4 microcontrollers leveraging ARMv7E-M ISA, and up to 10× better performance and energy efficiency compared to a baseline PULP system implemented in the same technology, paying an area overhead of only 17.5% and 4.1% with respect to the baseline core and cluster respectively.

The orders of magnitude improvements achieved with the proposed work compared with state-of-the-art MCUs demonstrate for the first time that software programmable edge inference of QNN models at ASIC-like efficiency is indeed possible on MCU-class devices, with an additional cost of area that is risible if compared to the one of dedicated accelerators. These results can be achieved by coupling architectural and power-aware micro-architectural design with leading-edge near-threshold FDX technology.

II. Related Work

Multi-precision low bit-width arithmetic is considered a well-established solution to deploy memory and power-hungry AI models at the extreme edge of IoT. It has been widely demonstrated that the precision of heavily quantized AI models is not significantly impacted in many IoT applications [7], [9]. Moreover, integer low bit-width arithmetic is advantageous at the edge of IoT for two reasons: it lowers the memory costs of the application, and it can reduce the latency and the energy of the computation if the underlying hardware supports low bit-width operations in an efficient way.

This scenario has motivated the design of specific arithmetic units to fulfill the AI requirements and improve the efficiency of the modern highly-quantized Neural Networks workload at the extreme edge of IoT. This trend impacts all the main categories of edge-AI computing platforms, usually grouped into three classes of devices: dedicated accelerators, FPGA solutions, and embedded Microcontroller (MCU) systems. We leave out GP-GPUs, certainly valuable for the Cloud Computing environment, but with a power envelope unaffordable for deeply embedded edge computing platforms working in a power envelope ranging from 10 mW to 100 mW. In this section, we recap the main state-of-the-art advancements in the computing arithmetic for AI as well as their use in each of the computing platform categories mentioned above, and we give insight on their applicability for DL deployment at the extreme-edge of the IoT.

A. Multi-Precision Arithmetic Units

The efficacy of low bit-width arithmetic architectures for QNNs workload has been widely demonstrated in the domain of dedicated accelerators. For example, in [15] the authors

[1] https://github.com/pulp-platform/pulp-nn.git

propose a bit-serial based MAC unit that operates on 1- to 16-bit multi-precision operand, while the second is always a single bit operand. The system is designed for the best efficiency, achieving a peak of 50.6 TOPS/W at a throughput of 184 GOPS. Another valuable example is [16], a DNN accelerator that embeds an energy-scalable multi-precision integer arithmetic unit and delivers 76 Gops/s with an efficiency of up to 10 Tops/s/W. Authors in [16] present a parallel Multiply-and-Add architecture based on the Booth-Wallace multipliers that can be reconfigured to perform 4b-to-16b $\times$ 16b operations. In the floating-point (FP) format domain, the authors present a MAC unit supporting reduced-precision FP16 and FP8 formats and also fixed-point arithmetic, achieving up to 75 TOPS/W efficiency.

Reduced FP formats have been widely explored also in the arithmetic units of GPUs, such as in the A100 Tensor Core by Nvidia [17]. The re-configurable architecture of the A100 can process FP64 formats (targeting High-Performance computing) down to the more efficient FP8, including the support for Brain Float 16b (BF16) and mixed-precision formats, specifically targeting Neural Networks. The A100 platform also supports integer arithmetic computation for inference tasks, with operands precision down to 4-bit (INT4).

While multi-precision arithmetic units are widely explored in ASIC solutions, few examples are presented in the domain of fully programmable edge devices, especially in the integer domain. GAP-8 [18], to deal with QNN workload, integrates into its architecture a dedicated CNN accelerator, which consists of a re-configurable arithmetic unit capable of supporting 8-bit $\times$ 8-bit or 16-bit $\times$ 4-bit operations. A different approach is shown in [19], where the authors present a re-configurable Parallel Balanced-Bit-Serial (PBBS) vector processing tile. It is suitable to improve the efficiency of sub-byte SIMD arithmetic operations of heavily leakage-dominated ultra-low-power design. However, the code serialization degrades heavily the performance in near- and super-threshold operating points.

B. Dedicated Accelerators

Dedicated accelerators are top-in-class for what concerns performance and energy efficiency on the QNN workloads. Having a highly specialized data-path, they can achieve performance in the order of 1 - 10 Gops/s with efficiency in the range of 10 - 100 Tops/s/W. A valuable example is Orlando [20], which reaches few TOPS/W of efficiency.

The arithmetic precision drop is a valuable technique to further improve the QNN efficiency also on ASIC accelerators [21]. UNPU [15] is an example of an accelerator supporting fully-variable weight bit-precision and capable of achieving a peak energy efficiency of 50.6 TOPS/W at a throughput of 184 GOPS. Moons et al. [16] presented ENVISION, an energy-scalable multi-precision DNN accelerator delivering 76 Gops/s with an efficiency of up to 10 Tops/s/W.

The high performance and energy efficiency achieved by these accelerators are counterbalanced by their poor flexibility, which makes the end-to-end deployment of real-sized DNNs harder. Moreover, even if modern dedicated architectures have a data-path somehow re-configurable (for example, allowing the execution of convolutions with different kernel sizes, 3x3, 5x5 or they provide the possibility to handle inception layers and/or residual connections), they can not be configured to support different kind of applications. In the IoT domain, instead, this flexibility is crucial. The DNN inference is usually only one part of a bigger application, where we additionally may want to handle peripherals, process the data through linear algebra, domain-to-domain transforms (even recurring to floating-point numbers), and manage the wireless transmission of the high-level compressed results. The poor flexibility and the high-cost per device make the ASIC solutions unattractive for their use as sensor-nodes at the extreme edge of the IoT.

C. FPGAs

The recent development of heterogeneous FPGAs such as the Xilinx Zynq family has enabled a higher level of flexibility to build CNN acceleration systems. Embedding general-purpose processors on the FPGA boards allows managing the program flow, handling the I/O sub-system, memory accesses, and communication, hence making easier to program the device and interact with external devices and sensors.

FPGAs usually come with DSP-capable hardware, but they have a power envelope in the Watt order. Thus the reduction of numerical precision for CNN models plays a key role in achieving good performance and energy efficiency. In the literature, we can find several FPGA-based solutions that exploit 16-bit fixed-point operands, such as in [22], but an ever-increasing number of works explore byte or sub-byte arithmetic. Qiu e al. [23] proposed a CNN accelerator supporting 8- and 4-bit data on a Xilinx Zynq board, while [24], [25] rely on ternary and binary networks. While most FPGA solutions feature a power envelope that can not meet the IoT end-nodes requirements, a new family of FPGAs announced by Lattice, namely Sense-AI [26], provide comprehensive hardware and software solutions for always-on artificial intelligence (AI) within a power budget between 1 mW and 1 W.

However, these ultra-low-power FPGA families feature limited LUTs capability and still are too expensive for many applications where MCUs are traditionally chosen thanks to their low cost. On the other hand, their efficiency remains way lower than what ASICs can offer. In addition, they can be reconfigured using a Hardware Description Language (HDL), increasing the productivity with respect to the above-mentioned ASIC solutions; still, their adoption remains an obstacle for the average IoT programmer, who demands for the highest flexibility of micro-controller systems.

D. Software Programmable solutions

Commercially available software-programmable general-purpose processors provide the highest flexibility for the deployment of the QNN at the extreme-edge. However, the ultra-low-power MCUs are not fast enough, if compared to ASICs, to sustain the QNN workload. Some programmable architectures, for example, exploit the computing power of multi-core processors, such as Raspberry Pi 3+ [27] or Pi 4, powered by a multi-core 64-bit System-on-Chip by ARM.

Other solutions couple a programmable core with a dedicated accelerator to improve performance. Notable examples are Kendryte [28], a dual-core RISC-V SoC outfitted with a CNN accelerator for AI applications, or more established devices like Movidius [29], or finally the Edge TPU [2] coupled with a CPU. Although these platforms are relatively inexpensive and flexible, their power consumption is too high as well. In the ultra-low-power domain, ARM proposed Trilium [30], a heterogeneous compute platform that provides flexible support for ML workloads. In many IoT applications, the cost constraints are very tight, and it is desirable to reduce area while having an MCU with boosted computing capabilities.

To enhance the performance of MCU systems, a recent effort by both academia and industry tries to extend them by either enriching their Instruction Set Architectures (ISAs) with custom instructions tailored for specific application domains or coupling the MCUs with ASIC accelerators.

ARMv7e provides SIMD instructions for 16-bit data, and the current generation of Cortex-M cores integrates this instruction set. Commercial embodiments of this ISA show a power envelope of few milliWatts, fitting the power budget of the IoT end-nodes. For example, STMicroelectronics proposed low-end (STM32L4 [3] family of microcontrollers, based on the Cortex-M4 cores) and high-end (STM32H7 [4] family embedding the Cortex-M7 cores) micro-controllers supporting DL processing at the edge. On the RISC-V side, the XpulpV2 ISA extensions [12] are meant for efficient digital signal processing, exploiting the SIMD paradigm down to 8-bit vector data. On top of this ISA, near-threshold multi-core heterogeneous platforms have been built to push the performance and the efficiency of QNN workloads. The commercially available GAP-8 [18] embeds a cluster of 8 RISC-V cores and a CNN-specialized accelerator that can give the MCU a 5 to 10× energy efficiency boost.

To provide these architectures with an efficient software back-end, the literature presents several solutions that achieves promising results on QNN workloads. It is worth citing the CMSIS-NN [10] library, developed by ARM to target their Cortex-M cores. As an additional contribution, this library has been extended to support heavily-quantized and mixed-precision kernels [14]. On the RISC-V side, PULP-NN [11] provides a solid back-end for RISC-V based multi-processors systems, supporting byte as well as sub-byte and mixed-precision QNN kernels.

Even if the sub-byte integer arithmetic is already adopted in training and quantization flows and ASIC/FPGA-based systems, it does not find enough room in the new generation of architectural solutions for MCU-based systems. The new generation of the ARM ISA for Cortex-M core [13], tailored for the QNN workload, features hardware loops, conditional execution instructions, and 8-bit SIMD instructions like the ones presented in [12]. However, it will not support lower-precision SIMD arithmetic.

Our work aims at bridging this ISA and hardware gap to improve the computing efficiency of heavily-quantized NN workloads at the extreme-edge of the IoT on fully-programmable MCU devices, nearing the level of specialization and energy efficiency of custom accelerators without forgoing flexibility. To this purpose, we extend the work presented in [31] and propose an energy-efficient multi-precision integer *Dotp* Unit, supporting SIMD vector operations on 16- down to 2-bit precision elements. To exploit the designed hardware in a fully programmable context, we integrate it into a parallel ultra-low-power (PULP) architecture of 8 RISC-V cores, extending their ISA with specific extensions consisting of low bit-width SIMD arithmetic operations and a family of *mac&load* instructions.

By exploiting the proposed SIMD *Dotp* Unit and its integration into a tightly coupled cluster of 8 processors, our contribution outperforms the state-of-the-art hardware and software solutions by at least two orders of magnitude in terms of performance and efficiency.

III. Background

A. Quantized Neural Networks (QNNs)

Quantized Neural Networks (QNNs) are the result of post-training quantization or quantization-aware training [32] procedures. After the quantization, each tensor $\mathbf{t}$ of the QNN (e.g., weights $\mathbf{w}$, input activations $\mathbf{x}$, or outputs $\mathbf{y}$) can assume only a finite set of values which are defined in a specific real-valued range $[\alpha_{\mathbf{t}}, \beta_{\mathbf{t}})$. These discretized real values can be mapped, through bijective functions, into pure integer numbers called *integer images* of the real-valued discretized tensors. More in detail the N-bit integer image $\widehat{\mathbf{t}}$ of the tensor $\mathbf{t}$ is connected to its real-valued quantized counterpart through the following function:

$$\mathbf{t} = \alpha_{\mathbf{t}} + \varepsilon_{\mathbf{t}} \cdot \widehat{\mathbf{t}}\,, \tag{1}$$

where $\varepsilon_{\mathbf{t}} = (\beta_{\mathbf{t}} - \alpha_{\mathbf{t}})/(2^N - 1)$. We call $\varepsilon_{\mathbf{t}}$ the *quantum* because it is the smallest amount that we can represent in the quantized tensor. Without loss of generality, we further constraint $\alpha_{\mathbf{x}} = \alpha_{\mathbf{y}} = 0$ for the input activations and the output features of each QNN layer. After mapping all the tensors in the integer domain, the application of the QNN operators (Linear Operator, Batch-Normalization, and the Quantization/Activation) can operate directly on the *integer images*:

$$\text{LIN}: \quad \varphi = \sum_n \mathbf{w}_{m,n}\mathbf{x}_n \iff \widehat{\varphi} = \sum_n \widehat{\mathbf{w}_{m,n}} \cdot \widehat{\mathbf{x}_n} \tag{2}$$

$$\text{BN}: \quad \varphi' = \kappa \cdot \varphi + \lambda \iff \widehat{\varphi}' = \widehat{\kappa} \cdot \widehat{\varphi} + \widehat{\lambda}\,. \tag{3}$$

In the LIN operator, the accumulator of the dot product operation will be represented, in general, with higher precision (e.g., 32 bits) with respect to the two inputs, since the quantum used to represent the accumulator $\widehat{\varphi}$ will be smaller than that of the two operands ($\varepsilon_{\varphi} = \varepsilon_{\mathbf{w}}\varepsilon_{\mathbf{x}}$). The same consideration also holds for the output of the Batch-Normalization operator. The final Quantization/Activation operator provides a non-linear activation semantic, which is essential for QNNs to work, and collapses the accumulator into a smaller desired bitwidth:

$$\text{QNT/ACT}: \quad \widehat{\mathbf{y}} = m \cdot \widehat{\varphi}' \gg d\,;\; m = \left\lfloor \frac{\varepsilon_{\varphi'} \cdot 2^d}{\varepsilon_{\mathbf{y}}} \right\rfloor . \tag{4}$$

[2] https://cloud.google.com/edge-tpu

[3] https://www.st.com/resource/en/datasheet/stm32l476je.pdf

[4] https://www.st.com/resource/en/datasheet/stm32h743bi.pdf

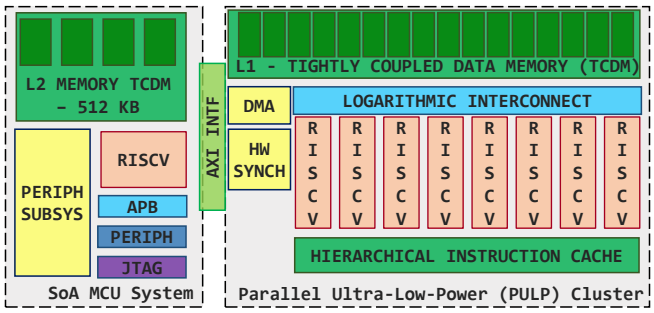


Fig. 1. The Parallel Ultra-Low-Power (PULP) system, consisting of a state-of-the-art microcontroller system accelerated by a parallel cluster of 8 RISC-V based processors.

d is an integer chosen during the quantization process in such a way that $\varepsilon_\varphi/\varepsilon_\mathbf{y}$ can be represented with sufficient accuracy inside m. The BN and QNT/ACT operators can also be implemented through a stair-case function by folding the BN and QNT/ACT parameters into a set of thresholds. The staircase-function compares φ with a set of 2^N thresholds to compress the result into N bits, with a computational complexity of $O(N)$. To implement the quantization with a thresholding-based method, we would need to store 2^N thresholds per output channel, which leads to a large memory footprint for real-world convolution kernels. Since the computational complexity is comparable between the two methods for real-world layers, we will always assume in the rest of the manuscript that the Quantization and Normalization steps are implemented with the BN and QNT/ACT operators, as explained in this section.

B. PULP cluster

Parallel Ultra-Low Power (PULP) is an open-source computing platform leveraging near-threshold computing to achieve high energy efficiency, leveraging parallelism to improve the performance degradation at low-voltage [33]. The PULP cluster we assume as a reference, depicted in Figure 1, is composed of eight RI5CY cores [12], each featuring a 4-stage in-order single-issue pipeline and implementing the RISC-V RV32IMCXpulpV2 Instruction Set Architecture (ISA). The XpulpV2 is a custom extension to the RISC-V ISA [12] meant for efficient digital signal processing computation. To this purpose, it includes hardware loops, post-modified access load and store instructions, as well as the support for SIMD operations down to 8-bit integer vector operands.

The cores of the baseline cluster synchronize through a shared Tightly Coupled Data Memory (TCDM) of 128 kB, divided on multiple-banks with a banking factor of two (i.e., 16 banks for the 8-cores configuration). The cores access the memory through a low latency logarithmic interconnect that serves the memory accesses in one cycle.

Meant to accelerate a microcontroller system, the PULP cluster communicates with its host through an AXI interface. It is also served with a DMA dedicated to the data transfers between the TCDM and the second level of memory, hosted by the microcontroller system, which also contains the program instructions for the cluster cores.

Each core fetches the instructions from a hierarchical instruction cache organized on two levels (the first private to each core, the second shared) to optimize the hit rate. The cluster is also provided with a Hardware Synchronization Unit that manages synchronization and thread dispatching, enabling low-overhead and fine-grained parallelism, thus high energy efficiency: each core waiting for a barrier is brought into a fully clock gated state.

C. QNN Execution Model

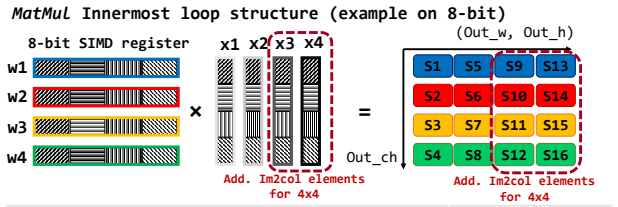
The software stack used in this paper to assess the results of the ISA extensions and the architectural explorations is derived from the open-source PULP-NN library[5], tailored for optimized parallel execution of QNN kernels on PULP and the above mentioned XpulpV2 ISA. In this work, we further extend the sub-byte symmetric convolution kernels of this library with the *Xpulpnn* ISA instructions.

The PULP-NN library relies on the Height-Width-Channel (HWC) data layout and on an execution flow optimized for resource-constrained microcontrollers. A convolution layer is implemented as a combination of three distinct phases.

im2col: This step takes the 3-D input activations in format (H,W,C) and, for a given output position, arranges its full receptive field along the filter and input channel dimensions into a 1-D vector. In this way, the full convolutional layer operation is converted into a scalar product between this vector and flattened weights. PULP-NN performs this operation for 2 output pixels concurrently, creating two distinct *im2col buffers* of $C_{in} \times F \times F$ elements each.

Matrix Multiplication: This step is the core of convolution, and it performs a sum-of-dot-product operation between the current *im2col buffer* and the sets of filters to produce higher precision results (the intermediate values of the output activations), which usually feature 32-bits. The kernel is highly optimized with the XpulpV2 instructions: hardware loops (*lp.setup*), load with post-increment (*p.lw*) and the SIMD *sdotp* (sum-of-dot-product) instruction which delivers 4 MACs in one cycle latency on 8-bit SIMD operands. As shown in Figure 2, to optimize performance the *MatMul* uses activations from two *im2col* buffers (associated to two spatially adjacent output pixels) and the quantized weights from four filter banks associated to four output channels. Exploiting the data locality within the RF enables the computation of eight output pixels per each iteration of the *MatMul* inner loop (4 channels $\times$ 2 adjacent pixels). This "4×2" structure is the result of a design space exploration aiming at finding the data reuse condition maximizing throughput [11]. Due to the limited amount of slots available in the RI5CY register file to store operands and accumulators, no further reuse can be exploited on RI5CY – in fact, wider *MatMul* structures (e.g., 4×4) would be detrimental as the additional accumulators would exceed the number of available registers, as visible from the Figure 2. This causes the compiler to continuously spill operands back and forth from the stack, introducing significant overhead [11];

[5]https://github.com/pulp-platform/pulp-nn



Register File Resources required	4x2 kernel	4x4 kernel
Loop Setup	2	2
Addresses for MEM access	6	8
MAC operands	6	8
MAC accumulators	8	16
Total	22	34 (does not fit)

(a) Layout and hardware resources (registers) of the "4×2" and the "4×4" layouts of the *MatMul* kernels.

Pseudo-assembly code of the 4x2 *Matmul*

```
      lp.setup      l1, l2,end     # HW loop setup
      p.lw          w1, 4(aw1!)    # Load with post-increment
      p.lw          w2, 4(aw2!)    ## of the address
      p.lw          w3, 4(aw3!)
      p.lw          w4, 4(aw4!)
      p.lw          x1, 4(ax1!)
      p.lw          x2, 4(ax2!)
      pv.sdotusp.b  s1, x1, w1     # 8-bit SIMD sum-of-dot-product
      pv.sdotusp.b  s2, x1, w2     ## instruction. One cycle latency
      pv.sdotusp.b  s3, x1, w3
      pv.sdotusp.b  s4, x1, w4
      pv.sdotusp.b  s5, x2, w1
      pv.sdotusp.b  s6, x2, w2
      pv.sdotusp.b  s7, x2, w3
end:  pv.sdotusp.b  s8, x2, w4
```

Wi Xi : weight/activation elements
AWi AXi : addresses for the MEM access
Si : accumulators
Li : loop setup

(b) Assembly code of the innermost loop of the "4×2" *MatMul* kernel . The figure puts in perspective the RF resources needed to run the loop.

Fig. 2. The Figure shows the structure and the assembly code of innermost loop of the *MatMul* kernel of the PULP-NN library. The "4×2" kernel structure fetches two activations (x_1 and x_2) from two different *im2col* buffers and the weights (w_1 to w_4) from four different filter sets to compute eight intermediate results (s_1 to s_8), requiring 22 registers available in the RF of RI5CY. The "4×4" layout can not be implemented on RI5CY, since the registers needed for the computation would not fit efficienctly the RI5CY register file.

***Quantization**:* As discussed in Section III-A, intermediate accumulators require 32-bit precision, and they need a final step of normalization and quantization to be represented in low bit-width form. These functions consist of one MAC operation, one shift, and one clip instruction per each accumulator to be quantized back into the desired precision. Contrarily to other quantization strategies such as thresholding-based quantization [14], the computational complexity of using explicit integer Batch-Normalization does not depend on the output activation precision. As a result, the *Quantization* stage adopted in this work will affect the overall performance to a higher degree when the precision is lower, and the *MatMul* is proportionally faster. Nevertheless, the advantage of this type of quantization resides in the lower memory footprint of its parameters with respect to thresholds. Per each output channel, we need to store 2^N thresholds for a final $N-$bit output activation. This behavior is more representative of the real-world QNN based tasks as shown, for example, in [34]. After normalization and quantization, the result is stored back into an 8-bit variable. For sub-byte operands, more output activations are compressed and stored back always into an 8-bit variable (which would contain either two 4-bit or four 2-bit elements) to reduce the memory footprint of the quantized output feature mapped.

IV. XpulpNN Extensions

This section presents the design of a high-efficient *Dot-Product* Unit supporting SIMD operations on vectors of 16b down to 2b elements. We integrate the unit into the RI5CY pipeline [12], and we extend its RISC-V ISA with a new set of extensions, namely *XpulpNN*, needed to effectively exploit the arithmetic unit. Then, we introduce the concept of the Mac&Load computation, presenting two different variants and comparing their benefits and their drawbacks. In the end, we integrate the RI5CY core extended with the new instructions into a parallel ultra-low-power cluster of eight processors, and we describe the software stack needed to execute the QNN convolution kernels on top of the *XpulpNN* ISA.

A. Multi-Precision Dot-Product Unit

The proposed *Multi-Precision Dot-Product* unit, depicted in Figure 3, computes the dot product operation between two SIMD registers and accumulates the partial results over a 32-bit scalar register through an adder tree, in one clock cycle of latency. The SIMD vectors are symmetric and can contain two 16-bit, four 8-bit, eight 4-bit, or sixteen 2-bit elements. We support the dotp operations interpreting the operands as signed or unsigned. Hence, we provide the inputs of the SIMD multipliers with an extra bit that sign- or zero-extends the actual single $N-$bit element of the SIMD vector. Therefore, each element is an $(N+1)-$bit signed word (Figure 3).

A common problem with an *N*-bit multiplier is that its output requires doubling the precision of the inputs (*2N*-bits) to cover the entire dynamic range of a multiplication operation. In some architectures, an intermediate register is used to store part of the multiplication result. In our case, being the elements of the SIMD vector 16- down to 2-bits, the *dotp* operations are implemented in hardware with a number of multipliers equal to the number of elements of the SIMD vector, followed by an adder tree that sums up the partial products, without any extra register to store the intermediate results. The stand-alone multiplier is designed to minimize the area-delay product, and it exploits a carry-save format without performing the carry propagation between different elements of the SIMD vector before the sum up phase performed by the adder tree. The sum-of-dot-product (*sdotp*) operation, which is the SIMD equivalent of a MAC operation, is supported by adding to the multipliers an additional 32-bit scalar operand at the input of each adder tree.

We integrate the *Dotp* unit into the pipeline of the RI5CY core, as depicted in Figure 4. The strategies examined during the design of the *Dotp* unit always consider such integration, optimizing the execution of dotp operations not only at the arithmetic level but also at the higher core-system level.

Our decision to replicate the hardware resources over different bitwidth dot product operations in the *Dotp* unit aims at minimizing the impact of the additional hardware on the critical path of the RI5CY core, which involves the path

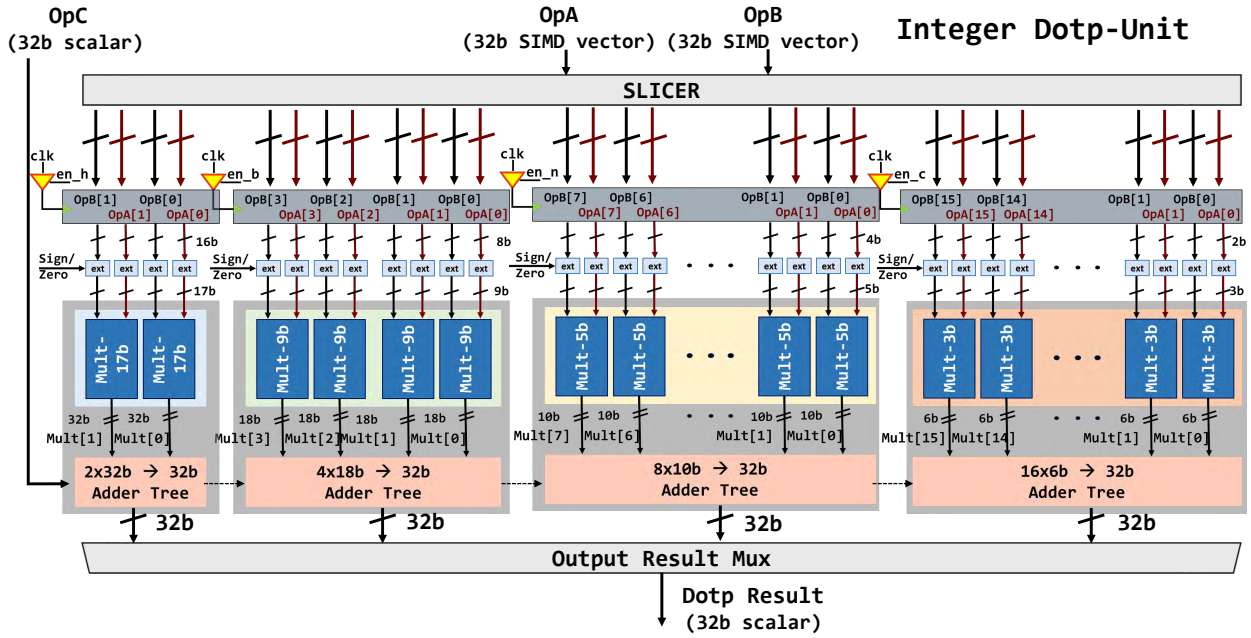


Fig. 3. Block diagram of the RI5CY Dot-Product Unit. To support the *XpulpNN* SIMD dotp-based operations, the 8×4 and the 16×2 SIMD MAC Units have been added. The figure includes the clock gating blocks needed to reduce the operand switching activity.

from the processor to the data memory and vice versa. The *dotp* operations are near to be timing critical since more logic is required with respect to a single-cycle multiplication operation due to the presence of the adder trees, needed to sum up all the partial products. Hence, sharing the multiplication resources among all the different bitwidth "regions", or even only sharing the adder tree to sum up over all the partial multiplication contributions, would be detrimental from the timing viewpoint: the additional combinatorial logic to select, split and distribute the operands and to enable the selected bit-width SIMD operation would have a negative impact on the overall speed.

The main drawback of our choice is in terms of area since we replicate hardware resources. As a direct consequence, the power consumption of the core system suffers a slight increase as well. To mitigate this effect on power consumption, we add a set of registers on the inputs of each bit-width region, and we perform clock gating to avoid switching for operands not involved in the current SIMD operation.

Despite a non-negligible impact on the total area of the EX-stage of the RI5CY core (18.4% of overhead with respect to the baseline EX-stage), the extended unit does not increase the critical path of the system, and it does not require pipeline stages in between the multiplication and the accumulation phases. Pipeline registers would result in execution stalls when computing back-to-back operations, introducing a huge overhead to the QNN workload, where most of the computation consists of sum-of-dot-product operations. Moreover, the dynamic power consumption of the core is kept almost unchanged thanks to our power-aware design, as shown in Section VI-A.

TABLE I
OVERVIEW OF *XpulpNN* INSTRUCTIONS FOR *nibble* (4-BIT) AND *crumb* (2-BIT) VECTOR OPERANDS. i IN THE TABLE REFERS TO THE INDEX IN THE VECTOR OPERAND ($i \in [0;7]$ FOR *nibble* AND $i \in [0;15]$ FOR *crumb*).

ALU SIMD Op.	**Description for *nibble***
pv.add[.sc].{n, c}	rD[i] = rs1[i] + rs2[i]
pv.sub[.sc].{n, c}	rD[i] = rs1[i] - rs2[i]
pv.avg(u)[.sc].{n, c}	rD[i] = (rs1[i] + rs2[i])>>1
Vector Comparison Op.	
pv.max(u)[.sc].{n, c}	rD[i] = rs1[i] > rs2[i] ? rs1[i] : rs2[i]
pv.min(u)[.sc].{n, c}	rD[i] = rs1[i] < rs2[i] ? rs1[i] : rs2[i]
Vector Shift Op.	
pv.srl[.sc].{n, c}	rD[i] = rs1[i] >> rs2[i] Shift is logical
pv.sra[.sc].{n, c}	rD[i] = rs1[i] >> rs2[i] Shift is arithmetic
pv.sll[.sc].{n, c}	rD[i] = rs1[i] << rs2[i]
Vector abs Op.	
pv.abs.{n, c}	rD[i] = rs1[i] < 0 ? -rs1[i] : rs1[i]
Dot Product Op.	
pv.dotup[.sc].{n, c}	rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7]
pv.dotusp[.sc].{n, c}	rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7]
pv.dotsp[.sc].{n, c}	rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7]
pv.sdotup[.sc].{n, c}	rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7] + rD
pv.sdotusp[.sc].{n, c}	rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7] + rD
pv.sdotsp[.sc].{n, c}	rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7] + rD

B. SIMD Extensions and Microarchitecture

To exploit the low bit-width integer SIMD computation enabled by the designed hardware, we extend the ISA of the target core with domain-specific instructions, namely *XpulpNN*. The proposed instructions, listed in Table I, extend the RV32IMCXpulpV2 ISA [12] with SIMD operations for 4-bit and 2-bit operands, namely *nibble* (indicated with *n*) and *crumb* (indicated as *c*) respectively, to improve the efficiency of low bit-width QNN kernels.

XpulpV2 supports three addressing variations: the first one uses two registers as source operands (*pv.instr.*{*b,h*}), the second variation uses one register and one immediate as

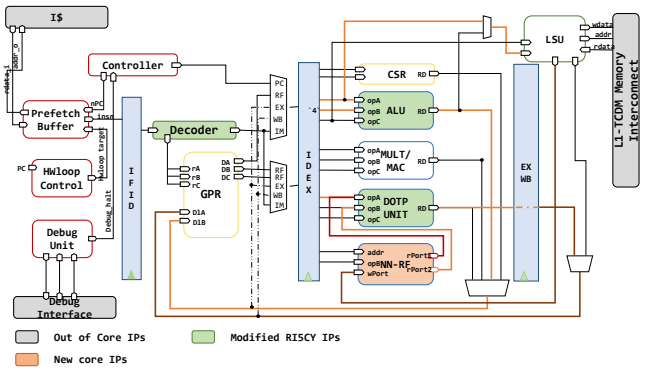


Fig. 4. The RI5CY pipeline. The Figure highlights the hardware blocks which extend the core micro-architecture to support the *XpulpNN* ISA.

source operands (*pv.instr.sci.*{*b,h*}), while the last one uses one register and replicates the scalar value in a register as the second operand for the SIMD operation (*pv.instr.sc.*{*b,h*}). Because of the limited room left in the encoding space of the baseline ISA, we propose the new *XpulpNN* crumb and nibble operations only in two addressing variants, and we do not implement the instruction format which uses an immediate value as the second operand (i.e., *pv.instr.sci.*{*b,h*}). Based on our experience, we argue that this choice is not a concern for the execution of QNN kernels: an immediate value can be stored in advance into a register without additional overhead.

The core of the *XpulpNN* ISA extension consists of the SIMD *dot product* instructions on packed vectors of 4-, 2-bit elements. The packed input registers can be interpreted as both signed or unsigned, or the first signed and the second unsigned. The accumulator, as well as the third scalar input in the sum-of-dot-product, can be either signed or unsigned. In addition to the *dot product* we support other SIMD instructions like maximum, minimum, and average for nibble and crumb packed operands, useful to speed-up the pooling layers and the activation layers based on the Rectified Linear Unit (ReLu) function. A group of arithmetic and logic operations (addition, subtraction, shift) completes the set of the *XpulpNN* SIMD instructions.

C. Compute&Update Instruction

In this section, we propose our hardware solution to further increase the speed-up of the QNN workload on RISC-V based pipelines. To perform a MAC operation or a SIMD *dot product* instruction on a RISC-based in-order single-issue processor, we first need to bring the two operands involved in the computation into the RF at the cost of two load operations. This means that only one-third of the executed instructions are relevant to the computation itself (i.e., the MAC instruction). We can formalize the concept defining the MAC operation efficiency (OPEF) metric that, in the case highlighted before, is equal to 0.33.

Since most of the QNN workload consists of MAC operations, we want that the OPEF is as high as possible to achieve high performance and efficiency, knowing that it cannot be higher than one on a single-issue processor (by construction). Data reuse at the RF level is an effective strategy to increase the OPEF of the MAC computation, as reported in [11] and already discussed in Section III-C. The innermost loop of the *MatMul* kernel of PULP-NN (Fig. 2) reuses two activations in the RF over 4 filters. This layout reduces the cost of the *sdotp* operations down to only six loads, bringing the OPEF to 0.57, with an improvement of 1.72 $\times$ compared to the baseline.

Our solution to improve the MAC efficiency even more without giving up the flexibility of a general-purpose RISC-V processor consists of the architectural and micro-architectural design of Mac&Load instructions, aiming at an OPEF close to 1. We explore two different designs of the Mac&Load operations for integration in *XpulpNN* and discuss their respective benefits and the drawbacks, aiming at the best trade-off between performance and implementation costs in terms of area, timing, and power consumption. To introduce the intuition at the basis of the Mac&Load paradigm, we discuss the assembly code of the *MatMul* kernel reported in Figure 2.(b). To hide the overhead of load operations, we propose to fuse the inner loop SIMD MAC (*pv.sdotp*) with the load within a single Mac&Load instruction. This is possible since the increment value (one word) is the same for all iterations, so it can be hardwired into the micro-architecture without being encoded into the instruction itself.

In the first design of the Mac&Load instruction, which we called Compute&Update ("C&U"), one of the operands of the Dotp Unit (e.g., one of the weights) is updated with a new memory element from the Load-Store Unit (LSU) of the core as soon as the SIMD MAC operation consumes it. The LSU accesses the memory location indicated by the "rs1" operand, as depicted in Figures 5.(a) and 5.(b). Afterward, the address consumed by the LSU is updated by one word in the ALU and stored back into the RF, similarly to the post-increment load of the *XpulpV2* ISA [12]. Data hazards, if any, are handled by stalling the pipeline exploiting the same signals of normal load instructions.

The RI5CY general-purpose RF (GP-RF) has two write ports, but the C&U instruction requires three accesses to store the output of the *dotp* operation, the updated address, and the new memory element. To avoid an additional cycle of latency, we would need to extend the GP-RF with one additional write port, which would be too expensive in terms of power and area. Our lightweight solution is therefore to provide the EX-stage of the core with a very small register file dedicated to this computation paradigm, namely the Neural Network Register File (NN-RF, as visible in Figure 5.(b)). The NN-RF is provided with one read port to feed the MAC unit with one operand and one write port to receive a new data word coming from the memory through the LSU. The NN-RF is sized in a way that all the loads related to the update of the weights in the innermost loop of the *MatMul* kernel are masked. From our exploration, the optimal number of registers is 4.

As visible from Figure 5.(a), the addressing of the NN-RF registers ("NN-RF[i]" field) is hard-encoded into the instruction to compress as much as possible all the necessary information to execute the C&U in the 32-bits of the encoding space. This causes the addition of four different C&U instructions, each one controlling one register of the NN-RF.

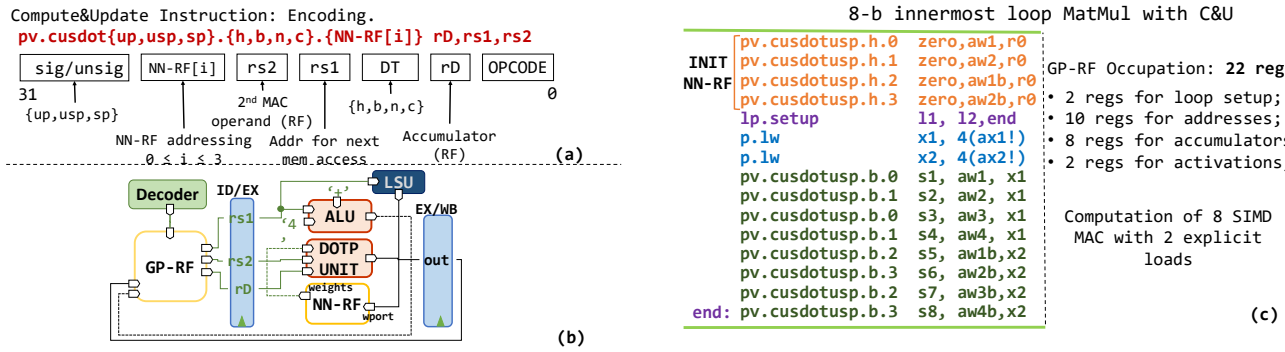


Fig. 5. In (a), the prototype of the Compute&Update (C&U) instruction is reported: the MSBs encode the interpretation of the operands, "NN-RF[i]" selects the current NN-RF register, "rs1" is the address for the next memory access, "rs2" is the second operand for the MAC unit, while DT encodes the data type of the operands (symmetric) and "rD" is the accumulator. In (b), we see the datapath to enable the C&U instruction. We add the NN-RF with one write port (connected to the LSU that fetches the new data accordingly to the "rs1" address) and one read port (multiplexed with the operand coming from the GP-RF) to feed the DOTP Unit. The ALU accepts the "rs1" operand to increment it by one word (" $+4$") and store it back to GP-RF. (c) depicts the innermost loop of the *MatMul* kernel. Before the loop, we need extra instructions to initialize the dedicated NN-RF registers that do not affect the performance. Inside the loop we occupy 22 regs of the GP-RF and reduce the load costs for the MAC down to 2 operations, bringing the OPEF to 0.8.

We added support for a C&U version of all the *sdotp* based instructions, interpreting the operands as signed/unsigned-signed/unsigned (*sp,usp,up*) and supporting 16-bit down to 2-bit SIMD operands (*h,b,n,c*).

To enable the MAC computation with one operand coming from the NN-RF, the Dot-Product unit is further modified by multiplexing its first operand coming from the GP-RF with the read port of the NN-RF (see Figure 5.(b)). Anytime the C&U instruction is issued in the EX-stage, the Dotp-Unit fetches its first operand (the weight element in the case of the PULP-NN *MatMul*) from the NN-RF. This micro-architecture enables the execution of the C&U instruction in one clock cycle of latency when the pipeline is fully operative and no stalls occur on the LSU-memory interface.

By replacing the *pv.sdotusp* instructions with the C&U equivalents in the innermost loop of the *MatMul* kernel, we are able to reduce the costs of explicit loads down to 2 with 8 SIMD MAC operations, as reported in Figure 5.(c) where we take as an example an 8-bit kernel. More in depth, we need some instructions of initialization to fill the NN-RF registers with the first operands involved in the MAC computations inside the loop. These few extra instructions do not affect the performance since they lay outside the critical loop. This implementation of the *MatMul* increases the OPEF to 0.8, further gaining a $1.40\times$ of improvement with respect to the original PULP-NN solution.

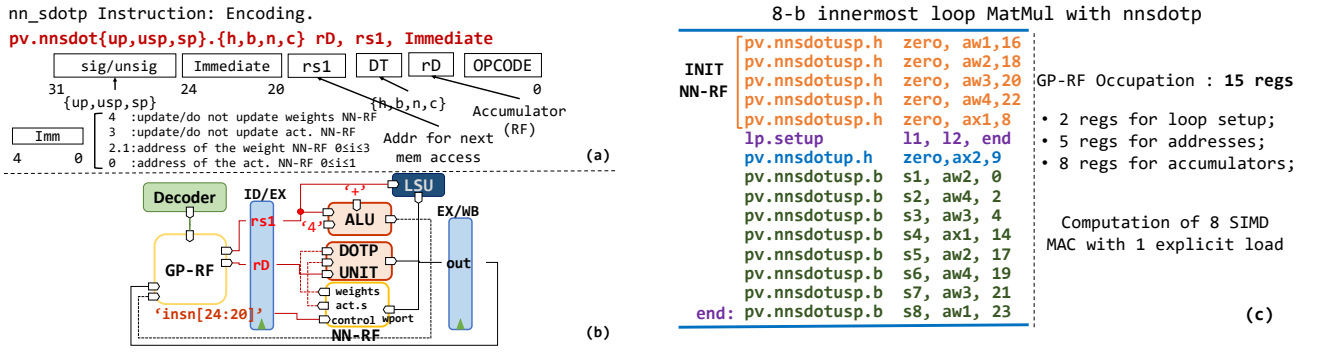
Despite the efficiency improvement achieved, we noticed some limitations related to the C&U operation. The main drawback is that we need to update the NN-RF register consumed with the MAC operation at each instruction execution. This is not a concern from a functional point of view since we are always able to mask all non-necessary loads into the fused instruction. However, the load operations are performed by the Load unit of the core, causing energy-expensive accesses to the memory and interconnect. In the context of tightly coupled shared-memory clusters, these additional loads create unnecessary contention, which degrades the overall performance.

Moreover, due to this "context-based" dependency, in the *MatMul* we need to use two different registers of the GP-RF to address the same weight location in the memory. If we refer to Figure 5.(c), the "aw1" address will be incremented by the *pv.cusdotusp.b.0* instruction by one word to fetch the next weight from the memory. The consumed and discarded weight is also needed in the computation with the "x2" activation element. To fetch the correct weight again, we must occupy another register, namely "aw1b".

The weakness is that we are not exploiting data locality on the weight elements anymore, and we are occupying redundant registers into the GP-RF. The number of occupied registers remains unchanged with respect to the *MatMul* of the PULP-NN library. Hence, also in this case, it is not possible to exploit the "4×4" *MatMul* data layout and its superior data reuse characteristics.

D. NN Sum-of-Dot-Product Instruction

The alternative version of the Mac&Load instruction we propose, namely "nn_*sum-of-dot-product* (nnsdotp)", overcomes the flexibility issues of the C&U presented above but requires more hardware resources to be integrated with the micro-architecture of the core. More in detail, we provide a solution that allows the operands stored into the NN-RF to be kept there as long as needed before being updated with the load operation of the fused *nnsdotp* instruction. This reduces the memory traffic, allows a higher grade of flexibility for data reuse (we are not limited by the compiler scheduler on the time we can keep an operand into the GP-RF), and solves the problem of using two different registers to encode the same address. The drawback of the *nnsdotp* is that the encoding of the new instruction is more complex. The functionality described above is encoded in a 5-bit Immediate field. This reduces the number of bits available to address another register of the GP-RF to feed the MAC unit with the second operand. Due to the regular structure of the *MatMul* though, this is not a concern at all. Rather, we can extend the NN-RF with two additional registers to host the two activation elements involved in the innermost loop computation of the *MatMul*. At the cost of a larger NN-RF compared to the solution



8-b innermost loop MatMul with nnsdotp

```
INIT NN-RF  pv.nnsdotusp.h  zero, aw1,16
            pv.nnsdotusp.h  zero, aw2,18
            pv.nnsdotusp.h  zero, aw3,20
            pv.nnsdotusp.h  zero, aw4,22
            pv.nnsdotusp.h  zero, ax1,8
            lp.setup        l1, l2, end
            pv.nnsdotup.h   zero,ax2,9
            pv.nnsdotusp.b  s1, aw2, 0
            pv.nnsdotusp.b  s2, aw4, 2
            pv.nnsdotusp.b  s3, aw3, 4
            pv.nnsdotusp.b  s4, ax1, 14
            pv.nnsdotusp.b  s5, aw2, 17
            pv.nnsdotusp.b  s6, aw4, 19
            pv.nnsdotusp.b  s7, aw3, 21
end:        pv.nnsdotusp.b  s8, aw1, 23
```

GP-RF Occupation : **15 regs**

- 2 regs for loop setup;
- 5 regs for addresses;
- 8 regs for accumulators;

Computation of 8 SIMD MAC with 1 explicit load

(c)

Fig. 6. (a) reports the encoding of the *nn_sdotp* instruction and describes the Immediate field. (b) depicts the micro-architecture design to support the instruction in the RI5CY pipeline. (c) shows the *MatMul* innermost loop implemented with the *nn_sdotp* instruction, highlighting the utilization of the GP-RF.

adopted with the C&U instruction, this solution guarantees more flexibility and performance.

As visible in Figure 6.(a), the 5-bit immediate addresses the NN-RF operands to be used in the current MAC operation: Bit 0 selects the activation register, bit 1&2 select the weight register, and bits 3&4 are set when we want to update either the addressed activation register or the weight register, respectively. Since we cannot update both weight and activation registers concurrently having a single LSU, these bits of the Immediate are mutually exclusive. To support this mechanism in hardware (see Figure 6.(b)), we provide the NN-RF with an additional read port that is multiplexed with the operand coming from the GP-RF to feed the Dotp Unit, as described above. Only when the *nnsdotp* instruction is issued, the Dotp Unit will receive both input operands from the NN-RF. The immediate bits act as control signals for the NN-RF.

The hardware cost of the *nnsdotp* instruction consists of the additional NN-RF with one write, two read ports, and some logic to distribute the operands to the Dotp-Unit. The arithmetic blocks are already present in the micro-architecture. Hence, the impact of both the Mac&Load instructions proposed is negligible in terms of the maximum frequency of the RI5CY core. From a power consumption point of view, the *nnsdotp* implementation has a non-negligible impact due to the additional NN-RF with two read ports and one write port. To avoid unnecessary switching activity when the *nnsdotp* is not executed, we perform operand isolation on the critical operands (e.g., at the input of the multiplexers of the Dotp Unit) and apply clock gating in the NN-RF block.

The implementation of the *MatMul* kernel using the *nnsdotp* instructions is reported in Figure 6.(c). Before entering the innermost loop of the *MatMul* we need to initialize all the NN-RF registers. In this case, contrarily to the previous kernel with the C&U instruction, we pay only one explicit load instruction to perform the same number of *dotp* instructions, increasing the OPEF up to 0.88, with an improvement of 1.1× with respect to the C&U case.

A major benefit of the kernel highlighted in Figure 6.(c) is that the occupancy of the GP-RF registers is reduced by 15 registers. This results by moving all the operands in the dedicated NN-RF, keeping the GP-RF free to host addresses for intermediate values and accumulators.

This condition leaves space for the implementation of the

4x4 *MatMul* layout, implemented using the nn_sdotp instruction.

```
INIT THE NN-RF  pv.nnsdotusp.h  zero, aw1,16
                pv.nnsdotusp.h  zero, aw2,18
                pv.nnsdotusp.h  zero, aw3,20
                pv.nnsdotusp.h  zero, aw4,22
                pv.nnsdotusp.h  zero, ax1,8
                lp.setup        l1, l2, end
                pv.nnsdotup.h   zero,ax2,9
                pv.nnsdotusp.b  s1, aw1, 0
                pv.nnsdotusp.b  s2, aw2, 2
                pv.nnsdotusp.b  s3, aw3, 4
                pv.nnsdotusp.b  s4, ax3, 14
(end):          pv.nnsdotusp.b  s5,  aw1, 1
                pv.nnsdotusp.b  s6,  aw2, 3
                pv.nnsdotusp.b  s7,  aw3, 5
                pv.nnsdotusp.b  s8,  ax4, 15
                pv.nnsdotusp.b  s9,  aw1, 0
                pv.nnsdotusp.b  s10, aw2, 2
                pv.nnsdotusp.b  s11, aw3, 4
                pv.nnsdotusp.b  s12, ax1, 14
                pv.nnsdotusp.b  s13, aw1, 17
                pv.nnsdotusp.b  s14, aw2, 19
                pv.nnsdotusp.b  s15, aw3, 21
                pv.nnsdotusp.b  s16, aw4, 23
```

Fig. 7. Detail of the "4×4" *MatMul* layout using the *nn_sdotp*. Storing the SIMD *sdotp* operands into the NN-RF reduces the pressure on the GP-RF. More room is left to host more accumulators. The assembly code shows how the innermost loop of the *MatMul* fit the register resources of the RI5CY core, thanks to the *nn_sdotp* instruction.

"4×4" *MatMul* structure. We need to fetch two additional elements from *im2col* memory buffers, whose addresses are stored into the GP-RF while the elements itself into the NN-RF. Reusing the weights also over the new activations, we can compute two additional pixels over four adjacent output channels (8 additional accumulators). Doing the math the occupancy of the GP-RF is of 32 registers (including the control registers for the HW loop), fitting the availability of the RI5CY GP-RF. This intuition is demonstrated by the implementation of the "4×4" kernel highlighted in Figure 7. Following exactly the same strategy as in the other cases with the initialization of the NN-RF, we pay a single load instruction to execute 16 *sdotp* operations, pushing the OPEF to 0.94, very close to the structural limit of 1.

To assess the benefits of the mac&load instructions at the micro-architecture level, we run simulations of the extended core executing multiple variants of the *MatMul* kernel: first using only the SIMD operations (*pv.sdotp*), and then using the C&U instruction and the *nn_sdotp* operation. For the latter case, also the optimized kernel layout is considered. Figure 8 reports the number of cycles required to perform a SIMD MAC operation (i.e., one *dotp* 8-bit operation counts as one MAC). As visible, the C&U improves the efficiency by 1.39 × with respect to the SIMD case. Thanks to the enhanced

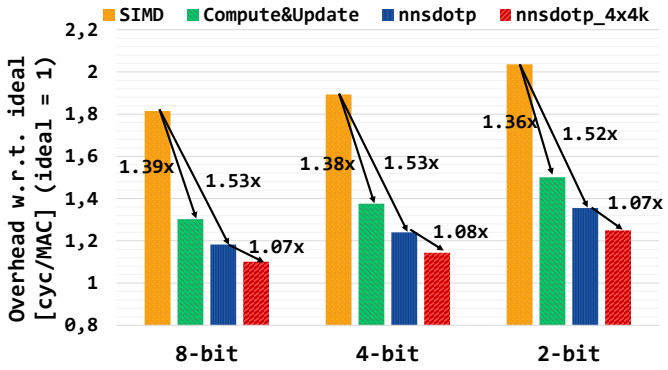


Fig. 8. Inverse of the Efficiency (lower is better) of the Matrix Multiplication kernel. The bar chart shows the cycles needed to the core to perform one SIMD MAC operation (4x8-bit. 8x4-bit, 16x2-bit respectively). The classical SIMD sdotp (XpulpNN) and the two versions of the MAC&Load instructions (macload, nn_custom) are considered. The nn_custom also allows to enlarge the Matrix Multiplication layout (nn_custom_4x4k).

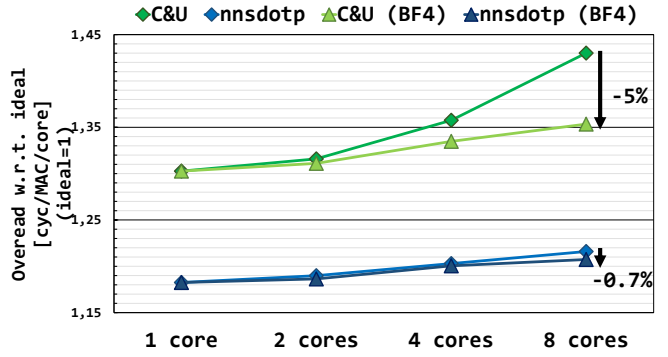


Fig. 9. Inverse of the MAC Operation Efficiency (lower is better) of the PULP cluster on 8-bit Matrix Multiplication (*MatMul*) kernels.

nn_sdotp instruction, after initializing the NN-RF registers, the innermost loop of the *MatMul* runs 1.10 × faster than in the C&U case and 1.53× faster than the SIMD case. Finally, optimizing also the *MatMul* layout, we gain an additional 1.07 × improvement with respect to the "4×2" layout and the *nn_sdotp*, with only 1.08 cyc/MAC, 1.65× higher than the SIMD case.

V. XpulpNN Integration

A. Cluster integration

After evaluating the improvement of the *XpulpNN* ISA on a single-core execution of the *MatMul* kernel, we integrate the extended RI5CY core into a PULP cluster of eight processors. Since the QNN workload is highly parallelizable, we expect a near-linear scaling of the performance when moving from single- to multi-core contexts [11]. We report in Figure 9 the results of the execution of the 8-bit *MatMul* kernel in terms of cycle needed by each core to execute a SIMD MAC operation, considering the execution of the kernel first with the C&U and then with the *nn_sdotp* instruction. The analysis carried out shows some drawbacks of the C&U instruction that limits the efficiency of the computation in a multi-core context. As visible from Figure 9, when executing the *MatMul* kernel with C&U on eight cores, its efficiency decreases with respect to the single-core execution. As described in Section IV-C, the C&U generates non-negligible traffic on the core-memory interface. This traffic results in many TCDM contentions in a multi-core context, causing each core to wait for the data from memory for more than one cycle. Splitting the L1 memory over more banks, we are able to partially limit this effect. More in detail, if we consider a banking factor of four ("BF4") (i.e., we double the baseline banking factor of two), the efficiency of the computation on eight cores increases by 5%, almost reaching the ones of the single core. However, this choice has a non-negligible impact on the power consumption of the system. Instead, the *nn_sdotp* does not suffer from this limitation, thanks to its capability to keep in the NN-RF one operand as long as we need, reducing the traffic on the core-memory interface when not needed. In a baseline configuration of the cluster (i.e., banking factor 2), the *nn_sdotp* reaches almost the same efficiency as in the "BF4" configuration.

B. Compiler & Parallel Programming Support

All the instructions of the *XpulpNN* ISA extensions can be inferred in the C code through the explicit invocation of built-in functions. In contrast with assembly inlining, this approach enables the lowering of built-ins into the high-level intermediate representation (IR) used by the compiler backend, allowing target-specific optimization passes to maximize the reuse of operands and efficiently schedule the instruction flow. This mechanism is essential to model the accesses to NN-RF consequent to Compute&Update semantic. Programmers do not have the visibility of the variables stored in NN-RF registers since their updates are hidden side effects from the C code perspective. The backend IR associated with the built-ins maintains track of these relations, and optimization passes take them into account.

This approach, of course, restricts the flexibility for the average embedded system programmer. However, our purpose is to expose the PULP-NN library functions as APIs. Practically, programmers never have to dig into a list of optimized low-level primitives, but they can select a library function (e.g., a convolution kernel). An example of this integration is in [34], where the backend library is integrated into a vertical QNNs deployment flow.

VI. Experimental Setup and Results

In this section, we evaluate *XpulpNN* both from a physical viewpoint, measuring and discussing the costs of the micro-architectural implementation in terms of area, power, and timing overheads with respect to the baseline RI5CY core and from a performance and energy efficiency perspective, comparing the execution of QNN workloads on top of the presented architectures with the State-of-the-Art Hardware and Software solutions.

To this purpose, we integrate both the RI5CY and the extended RI5CY cores into a Parallel Ultra-Low-Power (PULP) cluster of eight processors and perform a full implementation of the system in the Global Foundries 22nm FDX technology. We synthesize the two clusters with Synopsys Design Compiler-2018.3, and we perform a full place & route flow

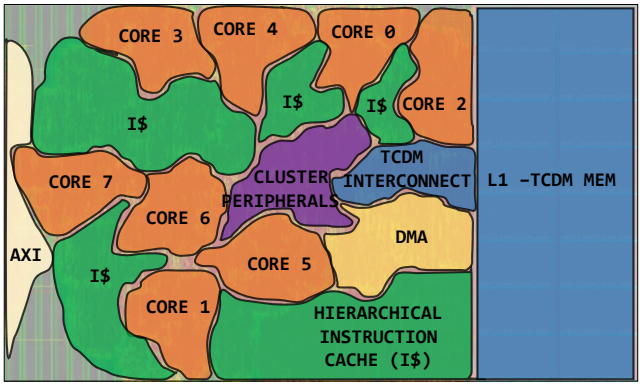


Fig. 10. Placed and routed design of the PULP cluster with eight extended RISCY cores, supporting the *XpulpNN* ISA.

using Cadence Innovus 17.11, in the worst-case corner (SS, 0.59V, $-40°$/ $125°$). The floorplan of the cluster is reported in Figure 10. The total area of the cluster and of the core and the timing results are obtained from layout measurements. To perform power overhead evaluations, we run timing-annotated post-layout simulations in the typical corner and in different operating points, targeting common QNN workloads as well as general-purpose applications. Thus, all the results presented in the following include the overheads (i.e., timing, area, power) caused by the clock tree implementation, accurate parasitic models extraction, cell sizing for setup fixing and delay buffers for hold fixing (neglecting these would cause significant underestimations in the clock tree dynamic power).

To compare our solution with the State-of-the-Art in terms of performance and energy efficiency, we benchmark a set of convolution layers. In the context of this work, we focus on the implementation of the PULP cluster since we target a parallel execution of the QNN workload. We assume then that the cluster is connected to a simulated micro-controller system that has the only duty of activating the cluster and hosts an L2 level of memory containing the application code. Since our goal is to improve the computing efficiency of the core kernels of a QNN inference task, we choose the layers such that their parameters fit the L1 memory of our systems to avoid additional overhead due to the memory transfers. However, the selected convolution layers are representative of the common tiles used in such types of devices to deploy QNN inference [34]. The benchmarked layers operate on a 16 $\times$ 16 $\times$ 32 input tensor with a filter size of 64 $\times$ 3×3×32 and on a 32×32×32 input tensor with a filter size of 64×3×3×32 respectively. As described in Section III-C, after the *MatMul* kernel, the intermediate results are compressed back into the desired precision through batch-normalization and activation functions.

A. Implementation Results

Table II shows a comparison between the RI5CY core and the extended RI5CY, implementing the *XpulpNN* ISA (with the mac&loadv2), in terms of area and power consumption, estimated on post-layout simulations of different applications. The total area of the extended RI5CY is $0.041mm^2$, with an overhead of 17.5% with respect to our baseline. Such increment is mostly due to the addition of the multipliers in the Dotp-Unit of the baseline core and of the extra-registers to build the NN-RF. The cluster area instead is of $1\ mm^2$ with the new core, 4% higher than the baseline. In Table II, we take into account also the cluster implementation with a banking factor of four to highlight the cost of this exploration in terms of area overhead. The cost of doubling the banking factor results in an additional area overhead of 4.2%. As introduced in Section IV-B, the duplication of the hardware resources into the Dotp-Unit allows us not to affect the critical path of the system. The maximum frequency achievable by both considered cores (RI5CY and the extended RI5CY) is the same.

Despite a non-negligible area overhead, the power consumption of the core is not affected significantly, as well as the power of the whole cluster system. To provide an accurate power estimation of the cores and characterize the whole system-level power consumption, we conduce post-layout power simulations in two different voltage corners:

TABLE II
AREA AND POWER CONSUMPTION RESULTS. WE CONSIDER TYPICAL AND WORST CASE CORNERS FOR EACH OPERATING POINT (HV= 0.8 V, LV=0.65 V). LIST OF CORNERS USED FOR IMPLEMENTATION: HV_TYP: TT, 25°C, 0.80 V; HV_SS: SS, 125°C/-40°C, 0.72 V; LV_TYP: TT, 25°C, 0.65 V; LV_SS: SS, 125°C/-40°C, 0.59 V. WE ALSO USE FAST CORNERS FOR HOLD FIXING. IN ALL CORNERS WE USE ALL PERMUTATIONS OF PARASITICS (CMIN/CMAX/RCMIN/RCMAX). **CORNERS USED FOR POWER ANALYSIS**: HV OP: TT, 25°C, 0.80 V, 660 MHz. LV OP: TT, 25°C, 0.65 V, 450 MHz.

Maximum Frequency [MHz] of the cluster with Ext. RI5CY cores				
	HV	**LV**	**HV_SS**	**LV_SS**
PULP Cluster	660	450	400	200
	RI5CY (baseline)		**Ext. RI5CY (with nn_sdotp)**	
Area [um2] (Overhead vs. baseline [%])				
Tot. Cluster	970856		1011254 (**4.1%**)	
Tot. Cluster (32 tcdm banks)	995210		1053446 (**5.9%**)	
Total Core	35131		41296 (**17.5%**)	
EX-Stage	13385		17744 (32.6%)	
Power Consumption of the CORE [mW] on an 8-b MatMul (Overhead vs. baseline [%])				
	HV	**LV**	**HV**	**LV**
Leak. Power	2.13	0.96	2.22	0.99
Dyn. Power	2.94	1.30	3.01	1.32
Tot. Power	3.05	1.35	3.12 (**2.1%**)	1.39 (**2.5%**)
Power Consumption of the CORE [mW] on a GP-application (Overhead vs. baseline [%])				
	HV	**LV**	**HV**	**LV**
Leak. Power	0.108	0.055	0.122	0.065
Dyn. Power	1.73	0.76	1.76 (1.7%)	0.78 (2.6%)
Tot. Power	1.84	0.82	1.88 (**2.17%**)	0.85 (**3.7%**)
Total Power Consumption of the PULP cluster [mW] (Overhead vs baseline [%])				
	HV	**LV**	**HV**	**LV**
MatMul 8-bit	41.8	19.3	41.6	19.3 (0.02%)
(with nn_sdotp)	–	–	43.7 (5.11%)	21.5 (11.5%)
MatMul 4-bit	–	–	35	16.1
(with nn_sdotp)	–	–	41.2	19
MatMul 2-bit	–	–	42.9	19.1
(with nn_sdotp)	–	–	48.9	24.1
GP Application	27.6	12.9	28.3 (**2.4%**)	13.3 (**3.1%**)

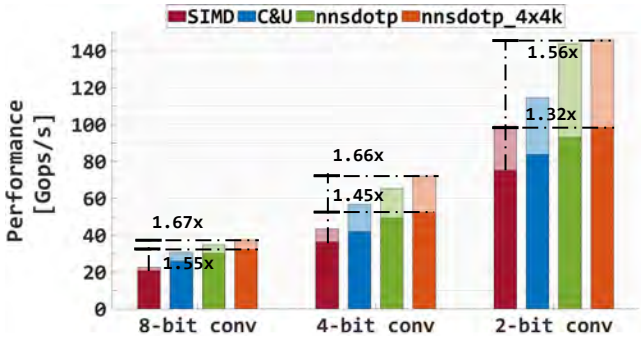


Fig. 11. Performance of the 8 core PULP clusters over different bit-width precision Convolution kernels, implemented with the instructions presented in this work. The lighted bars (higher-performance) refers to the *MatMul* kernel only, while the darker ones include also the quantization procedure (hence, the whole convolution). The cluster runs in the best performance operating point, at 660 MHz, 0.8 V in the typical corner.

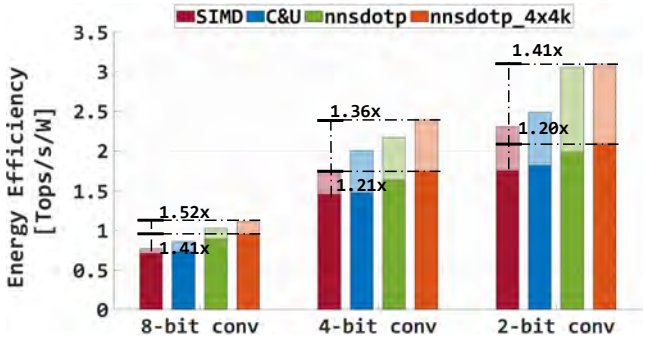


Fig. 12. Energy efficiency of the convolutions on the 8 core PULP clusters. The graph compares the solutions described in this work. The lighted bars (higher-performance) refers to the *MatMul* kernel only, while the darker ones include also the quantization procedure (hence, the whole convolution). The cluster runs in the best efficiency operating point, at 450 MHz, 0.65 V, in the typical corner.

the high-voltage corner (TT, 660 MHz, 0.80V) and the low-voltage one (TT, 450 MHz, 0.65V). We test 8-bit Dot-product based operations, the new nibble and crumb instructions, as well as the mac&load in its final version (nn_sdotp). Each kernel considered in the comparison is compiled with an extended GCC 7.1 toolchain that supports both *XpulpV2* and *XpulpNN* extensions. The Value Change Dump (VCD) traces are generated with Mentor Modelsim 10.7b and analyzed by Synopsys Prime Time 2019.12 to extract the power numbers. As visible in Table II, thanks to the clock gating techniques and to the operands isolation and despite the bigger core area, the extended RI5CY core runs an 8-bit Matrix Multiplication kernel (both the cores are using the 8-bit SIMD arithmetic instructions of the *XpulpV2* ISA) in almost the same power envelope of the baseline core, with a power overhead of only 3% in both considered corners. The same reasoning applies if we consider a General Purpose application, consisting of a mixture of the plain RISC-V ISA (RV32IMC) instructions such as load/stores, arithmetic, and control operations. This achievement is also visible at the system level, comparing the PULP cluster power consumption, demonstrating the light-weighted nature of the ISA extensions proposed in this work, and furthermore showing that we do not jeopardize the energy efficiency of the core on general-purpose benchmarks.

B. Benchmarking

To evaluate the performance and the energy efficiency gain achieved with the proposed *XpulpNN* extensions, we benchmark the convolution layers discussed above in different bit-width symmetric configurations (8-, 4-, and 2-bits). The kernels run on the extended RI5CY core, using different instructions of the *XpulpNN* ISA: classical SIMD operations, compute&update, nn_sdotp and the nn_sdotp optimizing the layout of the *MatMul*. This analysis aims at measuring the impact of the extensions on the whole convolution kernel of the PULP-NN library. The performance achieved, as well as the energy efficiency, are measured at the high-voltage corner (TT, 0.8 V, 25*C) and the low-voltage corner (TT, 0.65 V, 25°C) respectively of the post-layout simulations and reported in Figure 11 and 12 respectively. The peak performance and efficiency of the convolution layers are reached by implementing the *MatMul* kernel with the nn_sdotp instruction and an optimized 4×4 layout. In the 8-bit case, the improvement with respect to the classical SIMD implementation of the *MatMul* is 1.55× and 1.41× in terms of performance and efficiency, respectively. The little degradation of these two metrics compared to the ideal case where we consider only the execution of the *MatMul* kernel (bars in transparency in the Figure) is due to the quantization and compression of the intermediate *MatMul* results.

The impact of the quantization is much higher on the 4- and 2-bit convolution layers, especially when we refer to the optimized *MatMul* kernels. The reason for this behavior is that the computational cost for quantization does not depend on the bit-width of the compressed output feature map, meaning that it consists of the same operations no matter what is the precision of the final results. Considering the same layer parameters, the lower the precision of the *MatMul*, the less the iterations of the innermost loop (since in one *dotp* based operation we are actually performing 4, 8 or 16 effective MACs). Hence, the effective improvements in the *MatMul* kernel using the nn_sdotp instruction are mitigated by the batch-normalization and activation step on 4- and 2-bit convolution layers. As visible from the Figure 11, the performance improvement with respect to the classical SIMD implementation of the *MatMul* passes from 1.66× (1.56×) on the 4-bit (2-bit) *MatMul* itself to 1.45× (1.32×) on the whole 4-bit (2-bit) convolution layer. Obviously, these results directly translate into a corresponding degradation of energy efficiency. However, thanks to the optimized 4×4 *MatMul* kernel and the nn_sdotp instruction, we boost the convolution efficiency by up to 1.41× with respect to the SIMD implementation.

Despite the small degradation of performance and efficiency due to the quantization phases of sub-byte output activations, these cumulative improvements on the QNN kernels demonstrate the effective strategy of extending the ISA with domain-specific lightweight instructions to obtain high performance and energy efficiency on highly quantized QNN kernels, without affecting the system on other domain applications efficiency.

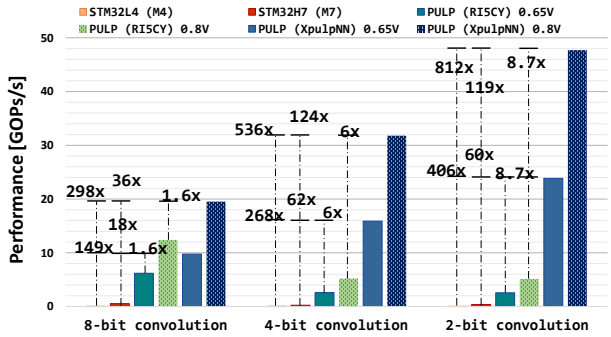


Fig. 13. The Figure shows the comparison of this work with the State-of-the-Art (high-end STM32H7 and low-end STM32L4 MCUs) and with the baseline RI5CY cluter, in terms of performance. The PULP clusters run in two operating points: high-voltage (0.8 V, 400 MHz) and low-voltage (0.65 V, 200 MHz). 8-, 4- and 2-bit simmetric convolution kernels are benchmarked to carry out the comparison.

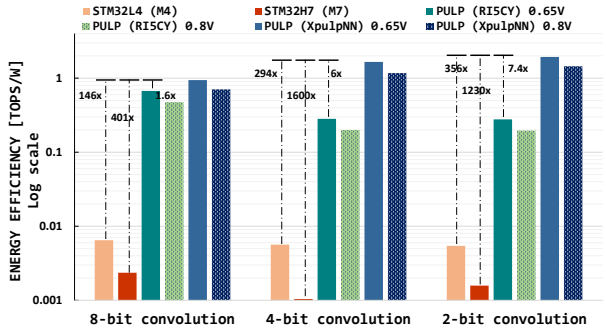


Fig. 14. Energy efficiency comparison of the this work with State-of-the-Art and the baseline RI5CY clusters. 8-, 4- and 2-bit simmetric convolution kernels are benchmarked to carry out the comparison.

C. Comparison With the State-of-the-Art

To put our achievement in perspective, we compare our results with state-of-the-art existing hardware and software solutions in terms of performance and energy efficiency. To carry out the comparison, we run the convolution layers on the RI5CY cluster using the PULP-NN library [11] and on two off-the-shelf STM32H7 and STM32L4 commercial microcontrollers previously introduced in Section II, using the extended CMSIS-NN library [14]. The performance and energy efficiency results are summarized in the Figure 13 and 14 respectively. For the implemented PULP cluster (with RI5CY and the extended RI5CY cores), we report two operating points: one at high-voltage, 0.8 V, 400 MHz and one at low-voltage, 0.65 V, 200 MHz, with the purpose to give insights on how much performance we trade-off with the energy efficiency at the highest voltage and vice versa. It is important to note that, since the STM32 MCUs are commercial products signed-off in the SS corners, the power analysis of our solution is carried out in the SS operating points (i.e., considering 400 MHz (200MHz) as the frequency for the best performance (efficiency) points) for a fair comparison. As visible from Figure 11, with the same operating condition, we improve the performance of the 4-bit (2-bit) convolution layers by 6× (8.7×) with respect to the RI5CY cluster. Thanks to the nn_sdotp we are also able to increase by 1.6× the performance on 8-bit convolutions. Almost the same grade of improvement is reached on the energy efficiency of such kernels, demonstrating that both clusters run almost in the same power envelope despite the enhanced ISA and the additional hardware. Also, the convolution kernels on the *XpulpNN* PULP cluster at the high(low)-voltage operating point run from 298× to 812× (149× to 406×) faster than the same kernels executing on the low-end STM32L4 using the CMSIS-NN library. In terms of energy efficiency, we outperform this microcontroller system by up to 356× in the best case (2-bit convolution, low-voltage operating point). Our performance gain with respect to the high-end Cortex-M7 based STM32H7 microcontroller is more limited than the previous case since the STM32H7 runs at 480 MHz and features a dual-issue core. In this case, we outperform its performance by up to 119×. Being a high-end microcontroller system, the STM32H7 suffers in terms of energy efficiency, where we do better by up to three orders of magnitude, as visible in Figure 14.

The presented results, coming out from the state-of-the-art comparison, are the consequence of the following insights: contrarily to ARM Cortex-M cores, the proposed solution has hardware support for 8-, 4- and 2-bit SIMD dotp-based operations and for the mac&load instruction. The STM32 based systems consist of a single-core chip, while our target architecture is a computing cluster of eight processors to improve the efficiency of the computation. The remaining performance/efficiency is gained due to the more scaled technology used to implement the PULP cluster compared to the one of the STM32L4 (90nm) and of the STM32H7 (40nm). In the end, the carried-out analysis shows for the first time that we can achieve ASIC-like energy efficiency on QNN workloads on fully programmable tiny MCU systems of the extreme-edge of the IoT. This outcome is obtainable by coupling the power-aware micro-architecture design and its integration in a multi-core computing cluster architecture with leading-edge near-threshold FDX technology.

VII. Conclusion

In this work we have presented a 2-bit to 16-bit multi-precision *Dotp* Unit that enables efficient computation of heavily QNN kernels at the extreme edge of IoT. We have integrated the unit into an open-source RISC-V processor, namely RI5CY, and we have performed optimizations at the core level to guarantee high energy efficiency in dotp-based computation. To exploit the designed hardware, we have provided the extended core with a set of low bit-width SIMD arithmetic extensions to the RISC-V ISA and we have shown the benefits at the ISA levels of implementing QNN kernels with the new proposed instructions. Furthermore, we integrated the new extended RI5CY core in a multi-core computing cluster, showing a near-linear speedup of the performance compared to the single-core execution. The implementation of the PULP cluster in leading-edge gf22 nm FD-SOI technology showed that: thanks to the design of a multi-precision low bit-width *Dotp* unit and the power-aware optimizations performed at the core level, the extended core does not jeopardize the efficiency

of RI5CY on general-purpose applications; given the same technology, the energy efficiency on byte and sub-byte kernels has been improved by up to one order of magnitude with respect to RI5CY. Our work shows at least two orders of magnitude improvements in performance and energy efficiency than state-of-the-art hardware and software solutions based on ARM Cortex-M cores. This scenario paves the way to software programmable QNN inference at the extreme edge of the IoT, promising ASIC-like efficiency with higher flexibility.

ACKNOWLEDGMENT

This work was supported in part by the EU Horizon 2020 Research and Innovation projects OPRECOMP (g.a. no. 732631) and WiPLASH (g.a. no. 863337) and by the ECSEL Horizon 2020 project AI4DI (g.a. no. 826060).

REFERENCES

[1] M. Hassanalieragh, A. Page, T. Soyata, G. Sharma, M. Aktas, G. Mateos, B. Kantarci, and S. Andreescu, "Health monitoring and management using internet-of-things (iot) sensing with cloud-based processing: Opportunities and challenges," in *2015 IEEE International Conference on Services Computing*. IEEE, 2015, pp. 285–292.

[2] C. A. Tokognon, B. Gao, G. Y. Tian, and Y. Yan, "Structural health monitoring framework based on internet of things: A survey," *IEEE Internet of Things Journal*, vol. 4, no. 3, pp. 619–635, 2017.

[3] S. Li, L. Da Xu, and S. Zhao, "5g internet of things: A survey," *Journal of Industrial Information Integration*, vol. 10, pp. 1–9, 2018.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[5] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and benchmarking of machine learning accelerators," *arXiv preprint arXiv:1908.11348*, 2019.

[6] D. C. Daly, L. C. Fujino, and K. C. Smith, "Through the looking glass-2020 edition: Trends in solid-state circuits from isscc," *IEEE Solid-State Circuits Magazine*, vol. 12, no. 1, pp. 8–24, 2020.

[7] 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.

[8] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 8612–8620.

[9] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

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

[11] 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.

[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, 2017.

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

[14] M. Rusci, A. Capotondi, F. Conti, and L. Benini, "Work-in-progress: Quantized nns as the definitive solution for inference on low-power arm mcus?" in *2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2018, pp. 1–2.

[15] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "Unpu: A 50.6 tops/w unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 218–220.

[16] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 246–247.

[17] Nvidia, "Nvidia a100 tensor core gpu architecture. available online: https://www.nvidia.com/content/dam/en-zz/solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf."

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

[19] B.-C. Wu and I.-C. 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, 2017.

[20] G. Desoli, N. Chawla, T. Boesch, S.-p. Singh, E. Guidetti, F. De Ambroggi, T. Majo, P. Zambotti, M. Ayodhyawasi, H. Singh *et al.*, "14.1 a 2.9 tops/w deep convolutional neural network soc in fd-soi 28nm for intelligent embedded systems," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 238–239.

[21] 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.

[22] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "An automatic rtl compiler for high-throughput fpga implementation of diverse deep convolutional neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

[23] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *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*, 2016, pp. 26–35.

[24] A. Prost-Boucle, A. Bourge, F. Pétrot, H. Alemdar, N. Caldwell, and V. Leroy, "Scalable high-performance architecture for convolutional ternary neural networks on fpga," in *27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–7.

[25] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 65–74.

[26] Lattice, "Lattice sensAI Delivers 10X Performance Boost for Low-Power, Smart IoT Devices at the Edge," https://www.latticesemi.com/About/Newsroom/PressReleases/2019/201911sensAI. Last accessed on Sept. 20.

[27] "Raspberry pi compute module 3+. 2019," https://www.raspberrypi.org/documentation/hardware/computemodule/datasheets/rpi_DATA_CM3plus_1p0.pdf.

[28] "2018. kendryte: K210 datasheet," https://s3.cn-north-1.amazonaws.com.cn/dl.kendryte.com/documents/kendryte_datasheet_20181011163248_en.pdf.

[29] "Intel neural compute stick 2. high performance, low power for ai inference." https://www.intel.com/content/dam/support/us/en/documents/boardsandkits/neural-compute-sticks/NCS2_Product-Brief-English.pdf.

[30] "Arm.project trillium machine learning platform," https://www.arm.com/products/silicon-ip-cpu/machine-learning/project-trillium.

[31] A. Garofalo, G. Tagliavini, F. Conti, D. Rossi, and L. Benini, "Xpulpnn: accelerating quantized neural networks on risc-v processors through isa extensions," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 186–191.

[32] 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.

[33] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "Mr. wolf: An energy-precision scalable parallel ultra low power soc for iot edge processing," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, 2019.

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



Angelo Garofalo received the B.Sc and M.Sc. degree in electronic engineering from the University of Bologna, Bologna, Italy, in 2016 and 2018 respectively. He is currently working toward his Ph.D. degree at DEI, University of Bologna, Bologna, Italy. His main research topic is Hardware-Software design of ultra-low power multiprocessor systems on chip. His research interests include Quantized Neural Networks, Hardware efficient Machine Learning, transprecision computing, and energy-efficient fully-programmable embedded architectures.



Giuseppe Tagliavini received the Ph.D. degree in electronic engineering from the University of Bologna, Bologna, Italy, in 2017. He is currently an Assistant Professor with the Department of Computer Science and Engineering (DISI) at the University of Bologna. He has co-authored over 30 papers in international conferences and journals. His research interests include parallel programming models, run-time optimization for multicore and many-core accelerators, and design of software stacks for emerging computing architectures.



Francesco Conti received the Ph.D. degree in electronic engineering from the University of Bologna, Italy, in 2016 where he currently holds an Assistant Professor position. His research focuses on advanced deep learning based intelligence on top of energy efficient programmable Systems-on-Chip – from both the hardware and software perspective. His research work has resulted in more than 40 publications in international conferences and journals and has been awarded several times, including the 2020 IEEE TCAS-I Darlington Best Paper Award.



Luca Benini holds the chair of digital Circuits and systems at ETHZ and is Full Professor at the Universita di Bologna. Dr. Benini's research interests are in energy-efficient computing systems design, from embedded to high-performance. He has published more than 1000 peer-reviewed papers and five books. He is a Fellow of the ACM and a member of the Academia Europaea. He is the recipient of the 2016 IEEE CAS Mac Van Valkenburg Award and the 2020 EDAA Achievement Award.



Davide Rossi Holds an Assistant Professor position at Università di Bologna. His research interests focus on energy efficient digital architectures in the domain of heterogeneous and reconfigurable multi and many-core systems on a chip. In these fields he has published more than 100 papers in international peer-reviewed conferences and journals. He is recipient of 2018 Donald O. Pederson Best Paper Award , 2020 IEEE TCAS Darlington Best Paper Award, 2020 IEEE TVLSI Prize Paper Award.