Alma Mater Studiorum - Università di Bologna

DOTTORATO DI RICERCA IN

INGEGNERIA ELETTRONICA, TELECOMUNICAZIONI E TECNOLOGIE DELL'INFORMAZIONE

Ciclo 34

Settore Concorsuale: 09/E3 - ELETTRONICA

Settore Scientifico Disciplinare: ING-INF/01 - ELETTRONICA

FLEXIBLE COMPUTING SYSTEMS FOR AI ACCELERATION AT THE EXTREME EDGE OF THE IOT

Presentata da: Angelo Garofalo

Coordinatore Dottorato

Aldo Romani

Supervisore

Luca Benini

Esame finale anno 2022

ALMA MATER STUDIORUM - UNIVERSITY OF BOLOGNA

Flexible Computing Systems for AI Acceleration at the Extreme Edge of the IoT

by

Angelo Garofalo

A thesis submitted for the degree of
Doctor of Philosophy

in the

Faculty of Engineering
Department of Electrical, Electronic and Information Engineering "G. Marconi (DEI)

May 2022

"Umuntu Ngumuntu Ngabantu."

"A person is a person through other people."

Zulu proverb.

Acknowledgements

I thank those who have dedicated their time to my training in these three years of PhD, contributing to my personal and professional growth.

In particular, I express my gratitude to the research group I am lucky and proud to be part of for the work done, for the experiences and the time we shared, as well as for the results obtained and the future opportunities.

With deep gratitude, I thank also my family for their support and closeness.

Lastly, certainly not least, thanks Francesca for everything you already know.

Abstract

Embedding intelligence in extreme edge devices allows distilling raw data acquired from sensors into actionable information, directly on IoT end-nodes. This computing paradigm, in which end-nodes no longer depend entirely on the Cloud, offers undeniable benefits, driving a large research area (TinyML) to deploy leading Machine Learning (ML) and Deep Learning (DL) algorithms on micro-controller class of devices. To fit the limited memory storage of these tiny platforms, full-precision Deep Neural Networks (DNNs) are compressed by representing their data down to byte and sub-byte formats, in the integer domain. The reduced footprint and limited accuracy drop, compared to full-precision models, make Quantized Neural Networks (QNNs) the natural target for TinyML. However, the current generation of micro-controller systems can barely cope with the computing complexity required by QNNs, motivating industry and academia to find new solutions to enrich these platforms with advanced computing capabilities.

This thesis tackles this challenge from many perspectives, presenting solutions both at software and hardware levels, exploiting parallelism, heterogeneity and software programmability to guarantee the highest flexibility and high energy-performance proportionality. The first contribution, PULP-NN, is an optimized software computing library for QNN inference on parallel ultra-low-power (PULP) clusters of RISC-V processors, showing more than one order of magnitude improvements, in performance and energy efficiency, compared to current State-of-the-Art (SoA) STM32 micro-controller systems (MCUs) based on ARM Cortex-M cores and running the CMSIS-NN library. However, when dealing with heavily quantized kernels, PULP-NN shows performance degradation due to the lack of low-bitwidth (sub-byte) computing capabilities at the hardware level. *XpulpNN*, the second contribution, aims to fill this gap by enriching the RISC-V Instruction Set Architecture (ISA) of the PULP cluster cores and their related micro-architecture with a set of domain-specific instructions. This silicon-tested solution achieves energy efficiency comparable with dedicated DNN accelerators and surpasses the efficiency of SoA ARM Cortex-M based MCUs, such as the low-end STM32M4 and the high-end STM32H7 devices, by up to three orders of magnitude. Intending to overcome the well known *Von Neumann bottleneck* while guaranteeing the highest flexibility, the final contribution integrates an Analog In-Memory Computing (AIMC) accelerator into the PULP cluster, creating a fully programmable heterogeneous fabric that demonstrates end-to-end inference capabilities of SoA MobileNetV2 models, showing two orders of magnitude performance improvements over current SoA analog/digital solutions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The last years have been characterized by a significant growth of the Internet of Things (IoT) interconnected devices [1], which pervade several application domains such as surveillance [2], agriculture [3], health monitoring [4, 5], structural health monitoring [6], robotics [7], automotive [8], industrial applications [9] and others [10]. This scenario requires the IoT end-nodes to acquire data from low-power sensors and send it wirelessly to the Cloud or other edge infrastructures, 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 [11].

The clear benefits of embedding the intelligence on IoT end-nodes have attracted the attention of a wide research area, referred to as Tiny Machine Learning (TinyML), 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 (e.g. Deep Neural Network (DNN), Convolutional Neural Network (CNN) models) that clash with the usual scarcity of computing and memory resources of deeply embedded systems, powered by batteries or energy harvesters. State-of-the-art DNN models feature floating-point high-precision arithmetic and typically run on General Purpose-Graphic Processing Unit (GP-GPU) and Field Programmable Gate Array (FPGA) devices in data centers. Unfortunately, these computing platforms are not usable in the

IoT environment, since they are characterized by a power envelope which is orders of magnitude higher than what is sustainable on extreme-edge devices.

However, one of the key characteristics of DNNs is their resiliency to strong arithmetic quantization in the integer domain. To reduce the size of the modern DNN topologies and make them fit the limited memory storage and computing capabilities of embedded low-powered devices like micro-controllers, recent progress in DL training methodologies has introduced novel quantization methods [12] that 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 [13–15]. Authors in [16], for example, show that the weights and the activations of a MobilenetV1 can be efficiently quantized to 8- or 4-bits with a loss on Top1 accuracy of only 0.8% and 3.2%, respectively, compared to the fully fixed-point precision. At the same time, this approach reduces the memory footprint by 4× (8-bits) and by 7× (4-bits).

The limited footprint and the good accuracy achieved make Quantized Neural Network (QNN) the natural target for TinyML. Moreover, the significantly higher computing efficiency that low-bitwidth integer arithmetic offers compared to the more costly floating point formats strongly motivates industry and academia to enable integer computing capabilities for Artificial Intelligence (AI) on top of micro-controller class of devices, addressing the challenge from both hardware and software perspectives.

The efficacy of low bit-width arithmetic architectures for the QNN workload has been widely demonstrated in the domain of dedicated accelerators, which are starting to gain attraction also for ultra-low power devices [17, 18]. However, these heavily specialized solutions, alone, are often not affordable in the extremely cost-conscious and fragmented IoT market, due to their high cost and their low flexibility. Solutions to offer higher flexibility are to couple Micro-Controller Units (MCU) with Application Specific Integrated Circuit (ASIC) [19–21], but the acceptance and penetration among TinyML application developers is still quite low.

MCUs are the standard IoT computing platforms, thanks to their flexible software programmability, low-cost and low-power characteristics. To enable QNN execution on MCU, optimized software libraries are presented in literature, often tailored on a target specific hardware to achieve reasonable performance and efficiency [22, 23]. However, none of the available solutions target the recent architectural template of parallel ultra-low-power platforms [24] that promise high energy-performance proportionality. On the hardware side, to empower MCUs with low-bitwdith integer computing, one must act on their Instruction Set Architecture (ISA). However, modern MCUs lack support at the 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.,

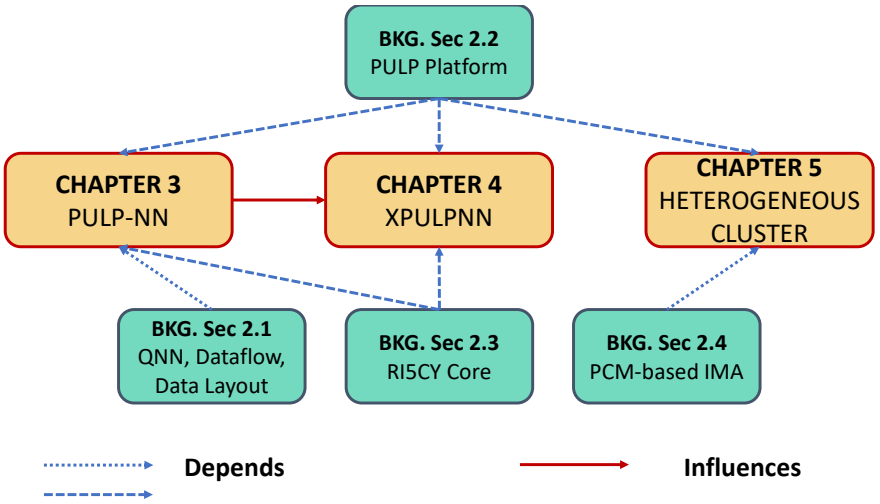


Figure 1.1: Thesis overview and dependencies between chapters and the background section.

ARMv7E-M) or 8-bits (e.g., RV32IMCXpulpV2 [25], ARMv8.1-M [26]) data. Hence, sub-byte quantization remains an effective technique to compress the footprint of DNN models on top of these devices [13], but it incurs in performance and energy overhead during the computation, as demonstrated in [23]: 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.

Recently, the emerging Analog In-Memory Computing (AIMC) paradigm promises outstanding efficiency on low-bitwidth Matrix-Vector Multiplication (MVM) operations, which are at the core of QNN routines. The IMC accelerators perform the MVM operations within the memory boundaries in the analog domain, overcoming the well-known memory bottleneck affecting traditional digital computing systems (the so called *Von Neumann bottleneck*) and achieving hundreds TOPS/W of energy efficiency. However, these platforms are not flexible to sustain the heterogeneity of the IoT workload. Hence, to operate at the extreme edge of the IoT they must be enclosed in heterogeneous programmable architectures, which raise new challenges at the system level still not investigated in depth by the research community.

1.1 Contributions

This thesis tackles the previously introduced challenges and state-of-the-art limitations from many perspectives, including both hardware and software designs and optimizations, targeting micro-controller class of devices. The most important contributions, and related publications, of this thesis are summarized in the following:

- The design of a open-sourced software library[1], namely *PULP-NN*, aiming at providing a set of optimized kernels to execute the inference of QNN on top of PULP based architectures, exploiting data-flow and data-layout structures tailored for MCU devices, as well as the computing characteristics and the parallelism of the underlying hardware to achieve high energy-performance proportionality. The contribution, discussed in Chapter 3, demonstrates significant improvements over the current state-of-the-art solutions [27–29]; on an end-to-end QNN inference task the proposed library, running on a commercial embodiment of the PULP platform, achieves one order of magnitude better energy efficiency with respect to state-of-the-art CMSIS-NN library [22], running on ARM Cortex-M based micro-controller systems such as the low-end (STM32L4 [30]) and high-end (STM32H7 [31]) devices;

- The design of a light-weight domain-specific set of RISC-V ISA instructions, namely *XpulpNN*, for multi-precision low-bitwidth integer computation, supported through the SIMD paradigm. The instructions are integrated into an existing RISC-V pipeline, designing the micro-architecture necessary to extend the datapath of the core to support the new ISA. The extended core is then integrated into a parallel cluster of 8 processors. The contribution, discussed in Chapter 4, aims at demonstrating the efficacy of the approach described here to empower MCU systems with AI computing capabilities without jeopardizing the power consumption of the core on general-purpose tasks. The experiments conduced show that this 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 [30] MCU and the high-end STM32H7 [31] MCU, on QNN tasks [32, 33]. The last part of Chapter 4 presents also a silicon prototype that includes contributions at the core level derived from the *XpulpNN* extensions, plus an additional hardware contribution at the system level: to save a significant energy factor when executing regular kernels like *Matrix Multiplications*, the cluster can be re-configured via software to operate either in classic Multiple-Instructions Multiple-Data (MIMD) mode or in SIMD mode (namely *Vector Lockstep Execution Mode (VLEM)*) [34].

- The design of a highly-heterogeneous computing architecture, including RISC-V general-purpose cores, a state-of-the-art in-memory computing accelerator and a specialized digital accelerator. This contribution, presented in Chapter 5, aims at demonstrating the efficacy of a highly-heterogeneous design approach to build new computing paradigm to tackle real-case challenges demonstrates end-to-end

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

inference capabilities on state-of-the-art neural network models, such as the *MobileNetV2*, within a power envelope typical of IoT devices. The solution proposed, on the inference task of a MobileNetV2, is one order of magnitude better in terms of execution latency than existing programmable architectures and two orders of magnitude better than state-of-the-art heterogeneous solutions integrating in-memory computing analog cores [35].

The rest of the thesis is structured as follows. In addition to Chapters 3, 4 and 5 that describe in details the main contributions of the dissertation, Chapter 2 provides information on background concepts necessary to better understand the key sections of the discussion. For clarity, the dependencies of each chapter on the concepts exposed in the background section and the chapters influence are depicted in Fig. 1.1. In the end, Chapter 6 ends the thesis with a summary of the contributions and draws the final conclusions.

1.2 List of Publications

The main contributions presented in this dissertation have been published in the following journal and conference papers:

- Angelo Garofalo, Manuele Rusci, Francesco Conti, Davide Rossi, and Luca Benini. Pulp-nn: A computing library for quantized neural network inference at the edge on risc-v based parallel ultra low power clusters. In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 33–36. IEEE, 2019

- Angelo Garofalo, Manuele Rusci, Francesco Conti, Davide Rossi, and Luca Benini. PULP-NN: accelerating quantized neural networks on parallel ultra-low-power RISC-V processors. *Philosophical Transactions of the Royal Society A*, 378(2164): 20190155, 2020

- Angelo Garofalo, Giuseppe Tagliavini, Francesco Conti, Davide Rossi, and Luca Benini. Xpulpnn: accelerating quantized neural networks on risc-v processors through isa extensions. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 186–191. IEEE, 2020

- Angelo Garofalo, Giuseppe Tagliavini, Francesco Conti, Luca Benini, and Davide Rossi. XpulpNN: Enabling Energy Efficient and Flexible Inference of Quantized Neural Networks on RISC-V based IoT End Nodes. *IEEE Transactions on Emerging Topics in Computing*, 2021

- Angelo Garofalo, Gianmarco Ottavi, Alfio di Mauro, Francesco Conti, Giuseppe Tagliavini, Luca Benini, and Davide Rossi. A 1.15 TOPS/W, 16-Cores Parallel Ultra-Low Power Cluster with 2b-to-32b Fully Flexible Bit-Precision and Vector Lockstep Execution Mode. In *ESSCIRC 2021-IEEE 47th European Solid State Circuits Conference (ESSCIRC)*, pages 267–270. IEEE, 2021
- Angelo Garofalo, Gianmarco Ottavi, Francesco Conti, Geethan Karunaratne, Irem Boybat, Luca Benini, and Davide Rossi. A Heterogeneous In-Memory Computing Cluster For Flexible End-to-End Inference of Real-World Deep Neural Networks. *arXiv*, 2022

The following publications with contributions by the author provide additional evidence and insights on the topics discussed in the thesis and are covered only in part by this dissertation:

- Alessio Burrello, Francesco Conti, Angelo Garofalo, Davide Rossi, and Luca Benini. Work-in-progress: Dory: lightweight memory hierarchy management for deep nn inference on iot endnodes. In *2019 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 1–2. IEEE, 2019
- Gianmarco Ottavi, Angelo Garofalo, Giuseppe Tagliavini, Francesco Conti, Luca Benini, and Davide Rossi. A mixed-precision RISC-V processor for extreme-edge DNN inference. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 512–517. IEEE, 2020
- Nazareno Bruschi, Angelo Garofalo, Francesco Conti, Giuseppe Tagliavini, and Davide Rossi. Enabling mixed-precision quantized neural networks in extreme-edge devices. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*, pages 217–220, 2020
- Alessio Burrello, Angelo Garofalo, Nazareno Bruschi, Giuseppe Tagliavini, Davide Rossi, and Francesco Conti. Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus. *IEEE Transactions on Computers*, 2021

Further publications with contributions by the authors not explicitly covered by this thesis are:

- Annachiara Ruospo, Riccardo Cantoro, Ernesto Sanchez, Pasquale Davide Schiavone, Angelo Garofalo, and Luca Benini. On-line Testing for Autonomous Systems driven by RISC-V Processor Design Verification. In *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2019

- Fabio Montagna, Stefan Mach, Simone Benatti, Angelo Garofalo, Gianmarco Ottavi, Luca Benini, Davide Rossi, and Giuseppe Tagliavini. A Low-Power Transprecision Floating-Point Cluster for Efficient Near-Sensor Data Analytics. *IEEE Transactions on Parallel and Distributed Systems*, 33(5):1038–1053, 2021

- Fabio Montagna, Giuseppe Tagliavini, Davide Rossi, Angelo Garofalo, and Luca Benini. Streamlining the OpenMP Programming Model on Ultra-Low-Power Multicore MCUs. In *International Conference on Architecture of Computing Systems*, pages 167–182. Springer, 2021

Chapter 2

Background

This chapter illustrates the background concepts necessary to introduce the contributions of this dissertation. Section 2.1 describes the Quantized Neural Networks, the dataflow schedule and the data layout adopted in Chapter 3 which presents the acceleration of QNNs through software optimizations, targeting ultra-low-power IoT edge devices.

All the contributions of the dissertation are built around the open-source PULP platform, introduced in Section 2.2. In Chapter 4 a set of domain-specific RISC-V ISA extensions is presented, whose micro-architecture extends the RI5CY core, described in Section 2.3. Section 2.4 describes the basic operations of the IMC array adopted in Chapter 5 to build an analog/digital heterogeneous computing cluster.

2.1 Quantized Neural Networks, Dataflow Schedule and Data Layout

QNNs are the result of post-training quantization or quantization-aware training [42] 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 (referred to also as INT-N) $\widehat{\mathbf{t}}$ of the tensor $\mathbf{t}$ is connected to its real-valued quantized counterpart through the following function:

$$\mathbf{t} = \alpha_{\mathbf{t}} + \varepsilon_{\mathbf{t}} \cdot \widehat{\mathbf{t}}\,, \tag{2.1}$$

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*:

$$\text{LIN}: \quad \varphi = \sum_n \mathbf{w}_{m,n}\mathbf{x}_n \iff \widehat{\varphi} = \sum_n \widehat{\mathbf{w}_{m,n}} \cdot \widehat{\mathbf{x}_n} \tag{2.2}$$

$$\text{BN}: \quad \varphi' = \kappa \cdot \varphi + \lambda \iff \widehat{\varphi}' = \widehat{\kappa} \cdot \widehat{\varphi} + \widehat{\lambda}\,. \tag{2.3}$$

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 QNN to work, and collapses the accumulator into a smaller desired bitwidth:

$$\text{QNT/ACT}: \quad \widehat{\mathbf{y}} = m \cdot \widehat{\varphi}' \gg d\,;\; m = \left\lfloor \frac{\varepsilon_{\varphi'} \cdot 2^d}{\varepsilon_{\mathbf{y}}} \right\rfloor\,. \tag{2.4}$$

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 thesis that the Quantization and Normalization steps are implemented with the BN and QNT/ACT operators, as explained in this section.

In this dissertation, we explore the case of INT-8, INT-4, INT-2 and INT-1 data types as they are the most natural ones to fit in a 32-bit register of the targeted MCUs.

The INT-1 format, where activation and weight values are expressed by binary values, is a special case because the convolution can be reduced to a logical XNOR and a bit-count operation:

$$\widehat{\varphi} = \text{popcount}(\widehat{\mathbf{w}} \,\text{xnor}\, \widehat{\mathbf{x}}) \tag{2.5}$$

where $popcount(\cdot)$ is the bitcount operator. Also in this scenario, a thresholding procedure is applied for compression.

On the model accuracy side, it has been demonstrated that, through specific retraining techniques, the accuracy drop-off of quantized fixed-point networks can be significantly reduced [13, 15, 16]. Choi et al. [42], for example, have proved that a 4-bit quantization leads to an accuracy level close to single-precision floating point representation. The accuracy drop is limited to 3% when running ResNet50 on Imagenet with 2-bit weights and 4-bit activations and to 6.5% when downscaling the weights and activations to 2 bits. Furthermore, the authors of [43] investigated the trade-off between energy efficiency and accuracy of QNN, highlighting the practical effectiveness of the sub-byte fixed-point networks. At the cost of specific retraining procedures, the accuracy drop of is kept very close to the single-precision floating point counterpart while the energy efficiency gain, at the iso-accuracy, is orders of magnitude higher. Moreover, for the investigated networks, trained on CIFAR-10 and MNIST datasets, the energy consumption achieved with 1- to 4-bit fixed-point networks, at iso-accuracy, outperforms the 8-bit counterpart by up to 10×.

A convolution layer, standing as the basic building block for a CNN or a QNN model, produces an output feature map based on a set of weight filters and the output from the previous layer. An activation value of any output feature map is computed as the dot product between a weights filter bank and a region of the input feature map, i.e. the C features values of every point under the area kw x kh of the filter. To efficiently implement this operation on an MCU-like device, the convolution is decomposed into two phases [22]: an *im2col* step to load the input features of the current convolution into a contiguous memory array and a dot product. Besides the memory requirements of the activation maps and the model parameters, the *im2col* demands an extra memory footprint of C x kw x kh values, on which the dot product operates. Fig 2.1(a) shows graphically this operation. Given this, the computation of one value of the output feature map, indicated as $O(m, x, y)$ becomes:

$$O(m, x, y) = \text{dot}\Big(W(m),\ im2col(x, y)\Big)\ , \tag{2.6}$$

where $W(m)$ is the m-th bank of weight filter, im2col is the unrolled input buffer of length C x kw x kh. The inner loop of the convolution dot product is realized through a matrix multiplication kernel, as depicted in Figure 2.1(b). In general, s output features

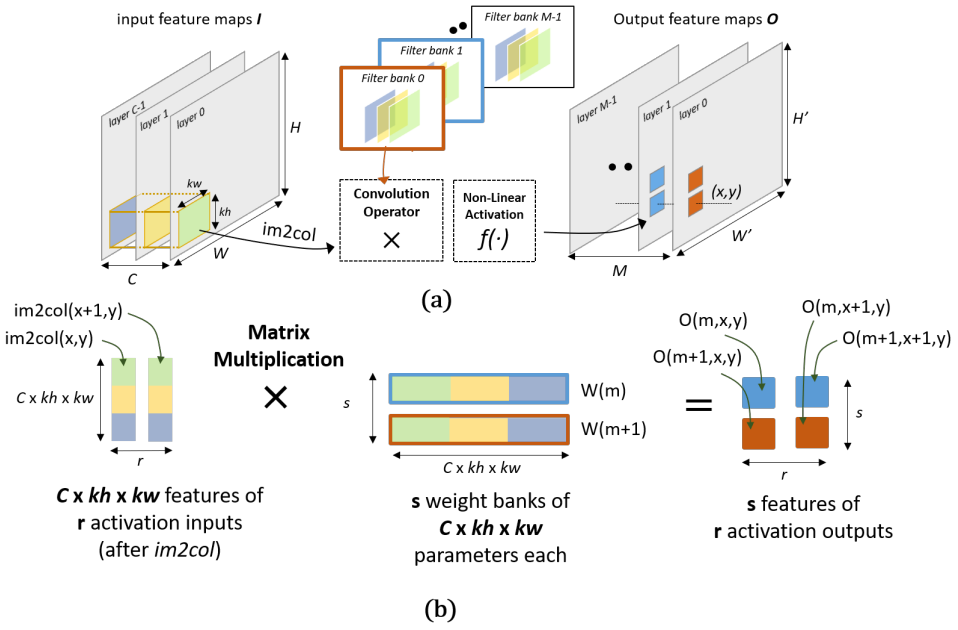


Figure 2.1: (a) Dataflow of the spatial convolution kernel (b) Convolution inner loop computation as a matrix multiplication.

of r activation outputs (s=2 and r=2 in the example in figure) can be computed at this low-level stage. As a specific case, CMSIS-NN implements a matrix multiplication kernel working on two spatially adjacent pixels of two consecutive channels inside the inner loop of the convolution kernel; we identify this configuration as 2×2, as explained in detail in Section 3.3.3.

Moreover, authors of [22] demonstrated the most convenient data layout to be Height-Width-Channel (HWC), as it introduces minor overhead when building the im2col buffer with respect to the Channel-Height-Width (CHW) layout. According to such a layout, the data along the channels is stored with a stride of 1, data along the width is stored with a stride equal to the number of channels C.

2.2 PULP Platform

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 [24]. The PULP cluster used as a reference in this dissertation is depicted in Figure 2.2. The computing cluster is composed of eight RI5CY cores [25], each featuring a 4-stage in-order single-issue pipeline and implementing the RISC-V

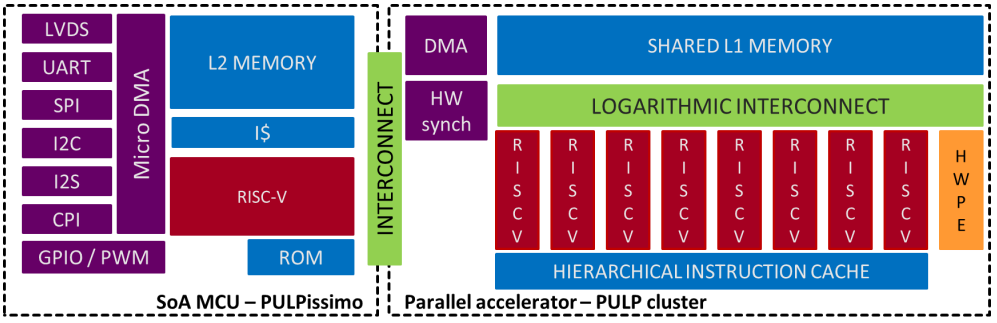


Figure 2.2: Overview of the PULP platform architecture.

RV32IMCXpulpV2 Instruction Set Architecture (ISA), meant to accelerate arithmetic intensive kernels and introduced more in depth in Section 2.3.

The cores of the baseline cluster communicate through a shared and word interleaved memory called Tightly Coupled Data Memory (TCDM), referred to as L1 memory. The size of the memory is parametrizable and can be divided on a number of banks which is also a design-time parameter. The cores access the memory through a low latency logarithmic interconnect (LIC), that serves the memory accesses in one cycle. The cluster workload can be also offloaded to accelerators, integrated into the cluster through a standardized interface [44], as shown in Fig. 5.1.

The cluster communicates with a micro-controller system, namely *PULPissimo*, that handles input/output peripherals, through an AXI interface. Moreover, it is served with a DMA controller dedicated to the data transfers between the TCDM and the second level of memory, hosted by the micro-controller 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 supported by a Hardware Synchronization Unit that manages synchronization and thread dispatching, enabling low-overhead and fine-grained parallelism, thus high energy efficiency: each core or accelerator waiting for a barrier, or more in general for a custom event, is brought into a fully clock gated state.

2.3 RI5CY Core

RI5CY is a 32-bit 4 stages pipeline in-order single issue processor [25], part of the PULP project [1]. Currently, it is maintained to industry standard by the non-profit global organization *OpenHW*, under the name of CV32E40P [2].

[1] https://github.com/pulp-platform
[2] https://github.com/openhwgroup/cv32e40p

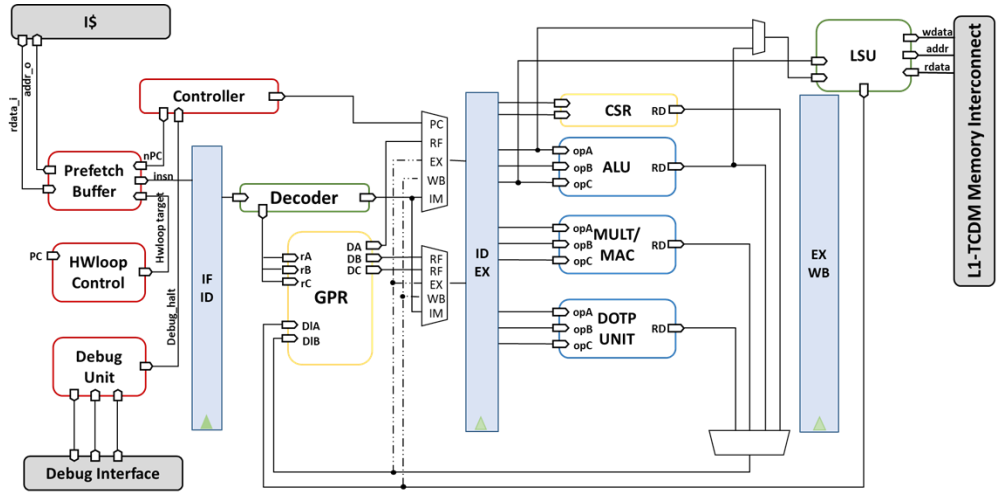


Figure 2.3: Diagram of the RI5CY pipeline.

The core implements the standard RISC-V ISA [45] with the I, M and C instructions. Optionally it supports the standard floating point instructions F. Besides the standard instructions, RI5CY supports the non-standard extensions called *XpulpV2* [25], that introduce several features to improve the efficiency of DNN inference kernels and, more in general, linear algebra and digital signal processing computation in the integer domain. Among the other useful operations, it is worth citing the support for hardware loops, post-modified access loads and stores, bit-manipulation instructions and support for SIMD operations down to 8-bit integer vector operands. Fig. 2.3 shows the diagram of the RI5CY pipeline.

2.4 PCM-based In-Memory Accelerator

In Chapter 5 the heterogeneous cluster is built around the IMC array presented in [46], which is based on a Phase-Change Memory (PCM) cross-bar. In this architecture, the memory devices are resistors with programmable conductance placed at the crosspoints of a 2D array with one terminal connected to horizontal wires called *word-lines* and the other terminal connected to vertical wires called *bit-lines*, enabling the execution of several computational primitives concurrently.

To perform the product of a matrix $\mathbf{A}$ by a vector $\mathbf{x}$, the PCM devices are programmed with conductance values proportional to the values $\mathbf{A}_{ij}$ of $\mathbf{A}$, with a precision of 4-bit (signed), as depicted in Fig. 2.4. Then the word-lines are driven with voltage pulses, whose duration are proportional to $\mathbf{x}_j$, using a set of digital-to-analog converters (DACs) with 8 bits of precision (signed). By Ohm's law, each PCM device contributes a current proportional to $\mathbf{A}_{ij} \cdot \mathbf{x}_j$ on the i-th bit-line, resulting in a total integrated current

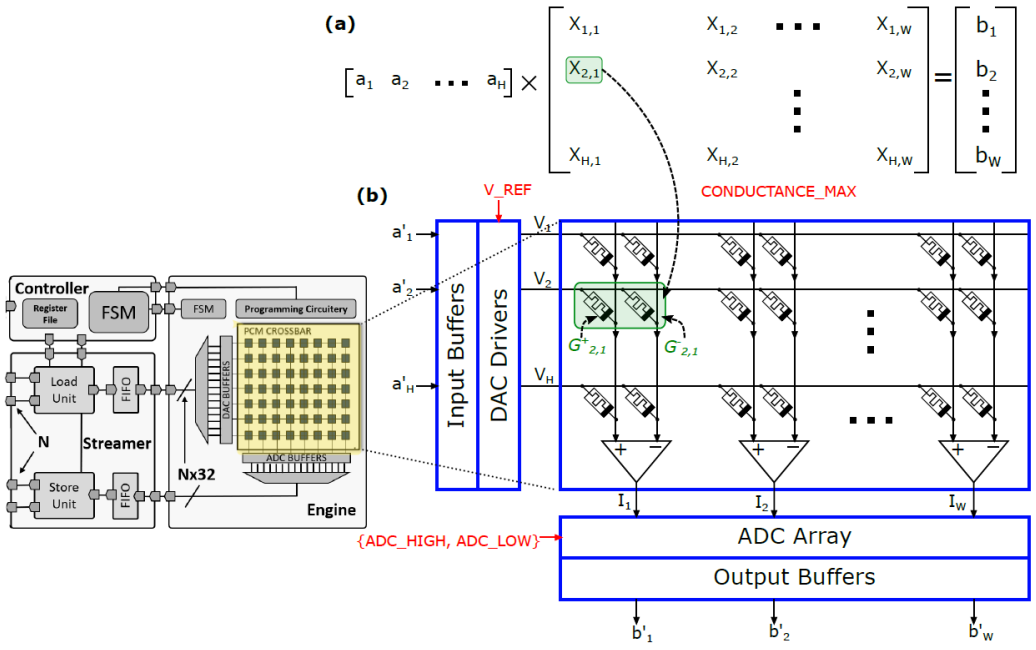


Figure 2.4: (a) Standard matrix vector multiplication where vector **a** is multiplied with matrix **X** to output vector **b** (b) Illustration of matrix vector multiplication operation on differential PCM crossbar array. It is reported also the integration of the IMC array within the HWPE, discussed in detail in Chapter 5.

proportional to the dot product $\mathbf{y}_i = \sum_j \mathbf{A}_{ij} \cdot \mathbf{x}_j$. At the end of each bit-line, there is an analog-to-digital converter (ADC) used to sample the bit-line current and convert it into an 8-bit digital value (signed).

For DNN inference, the **A** matrix can be used to store the weights of the linear part of a Fully Connected, Convolutional, or Depthwise Convolutional layer. Note that typically 2 PCM devices are used to denote a signed weight [47]. In conventional digital architectures, the dot product of 4-bit weights and 8-bit input activations requires a high-precision intermediate representation (often, 32 bits) that is subject to scaling, clipping, and quantization to produce a vector of 8-bit output activations [38]. In the IMC crossbar, instead, the intermediate representation is an analog current, while scaling, clipping, and quantization are performed directly by the bit-line ADCs by setting appropriate current limits.

Chapter 3

PULP-NN: QNN Acceleration on RISC-V IoT Processors

3.1 Introduction

3.1.1 Motivation

While efficient libraries for commercial MCUs have been proposed for edge QNN inference [22, 23], not many software solutions have been presented yet that efficiently exploit a parallel MCU architecture and offers supports for low bit-width computing kernels. This chapter fills this void by building the back-end library upon the recent architectural template of parallel ultra-low-power RISC-V based platforms such as GAP8 [20], which improve energy efficiency and performance in IoT edge devices coupling parallelism with low voltage operation [48].

3.1.2 Contribution

The main contributions are the following:

- The design of *PULP-NN* [1], an open-source optimized library based on the CMSIS-NN [22, 23] dataflow which includes a full set of kernels and utilities to support the inference of Quantized Neural Networks (8,4,2 and 1-bit) on a DSP-optimized RISC-V based processor. By fully exploiting the DSP extensions available within the ISA, it can achieve a speedup of 9× with respect to a plain*RV32IMC* ISA;

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

- The library optimizations for a Parallel Ultra-Low-Power (PULP) cluster of RISC-V processors, which leads to near-linear speedup with respect to single core execution, increasing the throughput of each kernel by up to 7.5× on eight cores;
- The optimization of the convolution kernel, the most computing intensive task of CNN workloads, by improving data reuse, with a further 20% performance gain with respect to the original kernel of CMSIS-NN [22], with a ∼1.9× improvement with respect to the GAP-8 NN native library and an overall efficiency of 49% in terms of MAC utilization, which implies just 1.01 LD/ST per MAC, and brings us to just a factor of 2 from the theoretical peak MAC utilization achievable using only register operands;
- The solution presented in this chapter is compared with State-of-the-Art architectures and software, by running a CIFAR-10 quantized model on the GAP8 8-core cluster, outperforming by 19.5× a high-end MCU (based on ARM CORTEX-M7) running the same network using the CMSIS-NN library. The inference with the proposed library also achieves 14.1× better energy efficiency with respect to a highly energy efficient MCU (based on ARM CORTEX-M4).

3.2 Related Work

The success of DL has paved the way to many different DL deployments on embedded computing platforms of all kinds. This section recaps the state-of-the-art and gives insights on its applicability to CNN inference at the extreme-edge, on IoT end-nodes.

3.2.1 Dedicated Accelerators for edge AI

3.2.1.1 Digital 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 [49], which reaches few TOPS/W of efficiency and Origami [50] that is capable of achieving a throughput of 274 Gop/s, with an efficiency of 803 Gop/s/W.

Dropping the arithmetic precision of CNN operands has demonstrated to be a useful technique to reduce the memory footprint and the energy cost for computation [43, 51–54]. UNPU [55] 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. [56] presented ENVISION, an energy-scalable multi-precision DNN accelerator delivering 76 Gops/s with an efficiency of up to 10 Tops/s/W. YodaNN [57] targets binary-weight networks and reaches energy efficiency up to 61 Top/s/W. Other accelerators exploit extreme quantization for the deployment of binary neural networks on silicon using in- or near-memory computing techniques (e.g., Brein [58], Conv-RAM [59]) with energy efficiencies in the range 20-55 Top/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.

3.2.1.2 Analog In-Memory Accelerators

A recent trend that leverages low-bitwidth computation is analog in-memory computing (AIMC) [60]. It overcomes the Von-Neumann bottleneck by executing the Matrix-Vector multiplication directly in-memory, reducing data movement and exploiting the high-parallelism of dense 2D memory arrays. Given that AIMC can only compute Matrix-Vector operations, several heterogeneous architectures have been proposed that adds digital electronics that perform the rest of the network (e.g., non-linear functions, residual layers, max-pooling, etc.). The attractiveness of these systems comes from the peak throughput and efficiency of DNN inferences. Examples are the works of Khaddam-Aljameh *et al.* [46] claiming 10.5 TOPS/W; Zhou *et al.* [61] with a peak efficiency of 112 TOPS/W; Jia *et al.* [62] peak efficiency of 30 TOPS/W.

Even though peak performance and efficiency of AIMC macros are outstanding, several fundamental challenges are still open to achieve the claimed levels in end-to-end applications: i) variability of analog computing can significantly impact the accuracy of the network; ii) need for specialized training; iii) poor flexibility, AIMC is well matched only

for a limited set of operations, mainly matrix-vector multiplication. Given these problems, most of these systems today have been demonstrated on small networks trained on simple data sets such as CIFAR-10 or MNIST [63].

Specialized architectures including both digital and analog accelerators can deliver remarkable performance and energy efficiency but lack flexibility. On the other hand, flexibility is a fundamental aspect when trying to accelerate DNN: neural networks are in continuous evolution, and the performance boosted thanks to an accelerator is only reachable for DNNs that can fit the target shape and size for which they have been designed.

3.2.2 FPGA based solutions

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 [64–67], but an ever-increasing number of works explore byte or sub-byte arithmetic. Qiu e al. [68] proposed a CNN accelerator supporting 8- and 4-bit data on a Xilinx Zynq board, while [69, 70] 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 [71], 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 FPGAs are currently too expensive for many applications where MCUs are traditionally chosen because of their low cost. 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.

3.2.3 Software Programmable Solutions

Commercially available software-programmable general-purpose processors provide the highest flexibility for the deployment of the QNN at the extreme-edge. While DNNs

Table 3.1: The table shows the trade-offs among the CNN computing platforms described in the related work section.

Summary of CNN Embedded Inference Computing Platform

	Performance	Energy Efficiency	Power Budget	Flexibility
ASICs [49, 50, 55, 57]	1-10 TOPS	10-100 TOPS/W	1 mW- 1 W	Low
FPGAs [64–68]	10-200 GOPS	1-10 GOPS/W	1-10 W	Medium
MCUs [30, 31]	100-300 MOPS	1-3 GOPS/W	1 mW- 1 W	High
PULP SoCs [20, 24, 76]	1-2 MOPS	30-50 GOPS/W	1-100 mW	High

are traditionally executed on programmable high-performance GP-GPU [72, 73] also with reduced precision support [74], these platforms are typically not designed to operate in the tight power envelope of IoT end-nodes, and their cost is off-spec too. Some architectures exploit the computing power of multi-core processors, such as Raspberry Pi 3+ [75], powered by a Quad-core ARM CORTEX-A53. Although these platforms are relatively inexpensive and flexible, their power consumption is too high as well.

To fit the power budget of IoT edge devices, many low power microcontrollers include ARM CORTEX-M cores. Among these solutions, STMicroelectronics produces low-power (STM32L4 family based on ARM CORTEX M-4 cores) and high-performance (STM32H7 family featuring ARM CORTEX M-7 cores) microcontrollers supporting DL processing at the edge [30, 31]. To improve the computing capabilities of such tiny and cheap computing platforms, ARM recently announced the development of the ARMv8.1-M [77] architecture, featuring Helium, an ISA extension tailored for DSP-oriented workloads, such as an inference task. However, such an extension is not supported yet by any device.

Other solutions move toward heterogeneous architectures, coupling microcontrollers with dedicated CNN accelerators, to deal with the extremely regular CNN workload. ARM proposed Trilium [19], a heterogeneous compute platform which provides flexible support for ML workloads. Conti et al. [44] proposed a convolution engine to be integrated in a microcontroller to speed up the convolutional kernels while Kendryte [21] is a dual-core RISC-V SoC outfitted with a CNN accelerator for AI applications. Flamand et al. proposed GAP8 [20], a multi-GOPS fully programmable RISC-V IoT-edge computing engine, featuring a cluster of 8 cores with dedicated DSP extensions and a CNN-specialized accelerator. These accelerators can give the MCU a 5 to 10× energy efficiency boost, but they are proprietary, closed, platform specific and currently not fully supported by the software design flows. Hence, their acceptance and penetration among application developers is still quite low. Table 3.1 summarizes the trade-offs

among the CNN computing platforms described so far. Next section will describe the State-of-the-Art of software solutions for MCU platforms, the main focus of this chapter.

3.2.4 Optimized Software Libraries

On the MCU side, the limited computational and memory capabilities make aggressive software and algorithmic optimizations necessary to deploy DNN inference models on them. An efficient solution to reduce DNN memory footprint is to use fixed-point arithmetic and quantization of both weights and activations into 8-bit or smaller data types, at the cost of a minor drop in accuracy [13, 16, 78]. Relying on fixed-point quantized networks, ARM proposed the CMSIS-NN library [22], which maximizes the performance of the QNN kernels on CORTEX-M series cores, supporting 16-bit and 8-bit fixed-point data. On the same trail, targeting a parallel MCU architecture such as GAP-8, Greenwaves Technologies released open-source a set of QNN kernels (16- and 8-bit data precisions) as part of a proprietary tiling solution [20]. The tiling procedure, exploiting the DMA controller available on GAP-8, hides the latency of fetching/storing activations and weights along the memory hierarchy introducing only a small overhead (a few %), thus enabling the processing of large networks whose layers may not fit the MCU on-board memory. This chapter focuses on the computational aspects of reduced precision quantized CNN inference. In this context, despite the demonstrated effectiveness of sub-byte aggressive quantization [43], only Rusci et al. [23] explored the inference speed as well as memory requirements of using low-precision (4-, 2- or 1-bit) convolution kernels on a Cortex-M7 microcontroller. The solution presented in this chapter aims at bridging this gap, leveraging the results of [43] and focusing on the computational side to enable efficient QNN inference at the edge on fully programmable devices. The solution presented outperforms the CMSIS-NN based solutions by one order of magnitude in terms of performance and energy efficiency.

3.3 PULP-NN

This section introduces the PULP-NN library and describes the optimization of the kernels with the presented RV32IMCXpulp extended ISA on a parallel cluster of eight processors and the optimization of the main computational kernel of the library: the matrix multiplication. We focus on the computational part since we are interested in exploring software solutions capable of achieving high computing performance and energy efficiency, on top of parallel edge architectures like PULP.

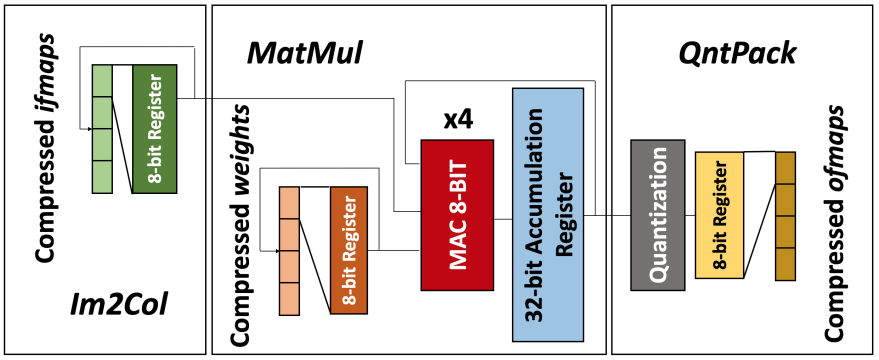


Figure 3.1: Concept scheme of the convolution kernel of the PULP-NN library.

3.3.1 Design and optimization on RISC-V

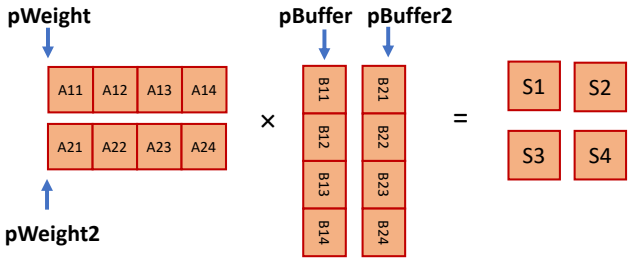
We present implementation details of the most significant QNN kernels on the target RV32IMCXpulp ISA. The experiments are conducted assuming that all the data resides in L1 memory of the PULP cluster.

INT-8 symmetric Kernels

This section focuses on the implementation details of the INT-8 convolution kernel, as it also provides a basis for the implementation of the INT-4 and INT-2 kernels. Starting from the implementation presented in Section 2.1, the INT-8 convolution consists of three phases, as depicted in Fig. 3.1: the *im2col* phase, the *MatMul* phase and the *QntPack* one.

The *im2col* step takes the 3-D input activations in the HWC format and, for a given output position, arranges its full receptive field along the filter and the input channel dimensions into a 1-D vector, the im2col buffer. In this way, the full convolutional layer is converted into a *MatMul* operation between this vector and flattened weights.

The structure of the *MatMul* kernel is 2×2, as discussed in Section 2.1. The following paragraphs give insights on how to optimize the kernel fully exploiting the target RV32IMCXpulp ISA. Since the matrix multiplication operation has to be looped over the size of each filter bank (C x kw x kh), the *hardware loops* provided by the target ISA are used to accelerate the *for* statement. In the inner loop, the load and store with post-increment is exploited since the access pattern to the im2col buffer and filter elements is extremely regular by construction. In the same way, the 8-bit SIMD instructions are used to work over more SIMD vector elements in parallel, to increase the throughput of the computation. Figure 3.2 graphically schematizes the execution of the inner loop of the *MatMul* kernel and reports the corresponding assembly code.



Pseudocode	Disassembled pseudocode	
For i=0 to size/4	lp.setup	// hardware loop
V4s A1 = *((v4s*) pWeight);	p.lw w1, 4(a0!)	// load with post-increment
V4s A2 = *((v4s*) pWeight2);	p.lw w2, 4(a1!)	
V4s B1 = *((v4s*) pBuffer);	p.lw x1, 4(a2!)	
V4s B2 = *((v4s*) pBuffer2);	p.lw x2, 4(a3!)	
S1 = sdotp4(A1,B1);	pv.sdotsp.b s1, w1, x1	// 4x8bit vectorial sdotp
S2 = sdotp4 (A1, B2);	pv.sdotsp.b s2, w1, x2	
S3 = sdotp4(A2, B1);	pv.sdotsp.b s3, w2, x1	
S4 = sdotp4(A2, B2);	pv.sdotsp.b s4, w2, x2	
end	end	

Figure 3.2: 2×2 sized matrix multiplication kernel for INT-8 data operands.

After filling two im2col buffers that are needed to compute two spatially adjacent output pixels during the *im2col* phase, the *MatMul* inner loop takes place as follows. At every iteration of the loop, four consecutive elements are loaded into the register file from each of the two im2col buffers (pointers *pBuffer1* and *pBuffer2* in the figure), and from two weight banks (pointers *pWeight* and *pWeight2*), after casting INT-8 pointers to *v4s*. The total number of load operations required is four. In this way we have sufficient elements to set four *sdotp4* built-in functions over four different accumulators. Hence, in a single run of the inner loop of the matrix multiplication kernel, we can compute four sdotp4 instructions, which correspond to 16 MAC operations, at the cost of four load instructions.

Since the fully connected kernel is a simple MVM, the previous methodology naturally scales to it. Here there is no need to build the im2col buffer since the spatial dimension of the filters is the same size as the spatial dimension of the input feature map.

To reduce load instructions and exploit a data reuse mechanism, the fully connected kernel implements $2x1$ matrix multiplication kernel within the inner loop (see Section 3.3.3 and Figure 3.7). By loading two different subsets of weights, two consecutive output pixels along the channel dimension can be computed in parallel. By using the SIMD ISA extensions as before, only three loads are required to set two *sdotp4* vector operations per loop cycle, which translates in 8 MACs.

The results of the *MatMul* kernel are 32-bit long, since the accumulator features a precision higher than operands, as described in Section 2.1. A final step of normalization

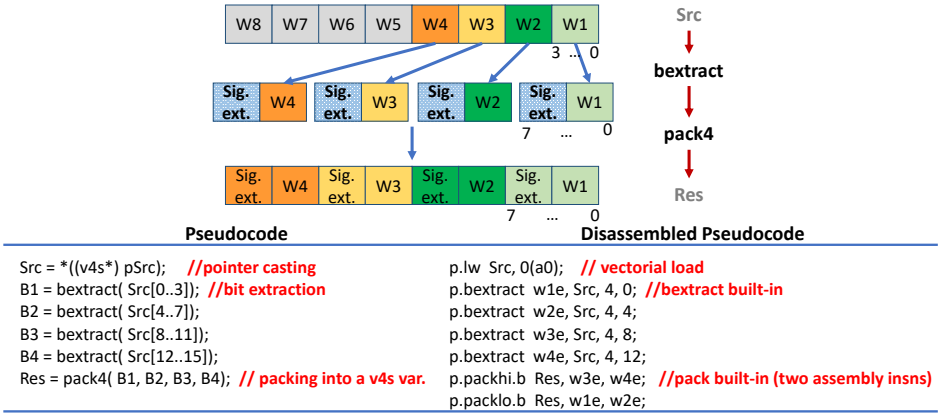


Figure 3.3: INT-4 to INT-8 unpacking function.

and quantization, namely *QntPack*, is thus needed to bring back the intermediate result in low bit-width form (INT-8 in the example considered). For INT-8 output features inexpensive operations such as shifting and clipping operations can be used, using ISA-level instructions. In the sub-byte cases, the *QntPack* includes also additional *packing* functions that will be discussed in the dedicated section.

Ancillary operations also take benefit of the DSP extensions. ReLU, which consists of a simple max looped over the input feature map, exploits *hardware loops*, load store with post-increment and the SIMD *max4* built-in instruction. The same is also used to optimize the max-pooling kernel, which is implemented in two steps: first along the width dimension, working destructively *in situ* on the input buffer; then along the height dimension.

Sub-byte and Mixed-Precision Support

The smallest data type well supported by the ISA with the SIMD extensions is INT-8. To exploit efficiently such vector operations, it is necessary to provide additional support functions to convert sub-byte data, i.e. INT-2 and INT-4, into INT-8. Having sub-byte operands compactly stored in memory, in the case of INT-4 data two consecutive elements are placed in a single byte. The casting operation, realized through the *pulp_nn_int4_to_int8* function, takes place either when building the im2col buffer as well as in the innermost loop of the matrix multiplication kernel to "unpack" weight elements. To reduce the overhead due to the unpacking operations, combined use of the *bextract* and *pack4* built-in functions allows to extract four INT-4 elements (weights or pixels) with few instructions, as shown in Figure 3.3. After loading eight INT-4 data with a single load, four elements are extracted by means of the *bitextract* built-in and packed into one single SIMD *v4s* variable, which feeds the matrix multiplication kernel.

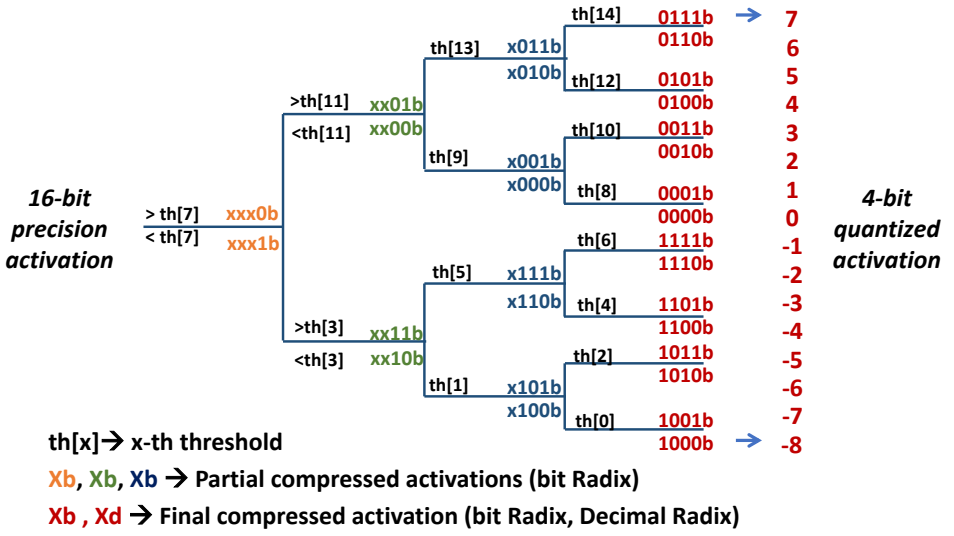


Figure 3.4: Binary tree implementation of the staircase compression function for 4-bit operands and iterative construction of the result.

On the same trail of INT-8 kernels, also in this case a final *QntPack* step is required to restore the 16-bit long intermediate *MatMul* results into the low bit-width representation range (INT-4 in the use case considered). The difference with respect to the INT-8 case is that for sub-byte kernels the *QntPack* step is more complex and consists of an optimal balanced binary tree function, named *pulp_nn_int4_quant* that at each node compares the 16-bit intermediate accumulator with one of the corresponding $2^4 - 1$ threshold values, as shown in Fig. 3.4 for the INT-4 case [2]. To save memory footprint, two consecutive output INT-4 data are stored in a single-byte variable using additional *packing* functions after the thresholding-based quantization. This is implemented efficiently, exploiting the *bitinsert* built-in function that acts as a natural counterpart of the *bitextract*, which compresses the data and packs them into INT-8 variables. A graphical explanation of the compression mechanism is provided in Figure 3.5. A similar process is implemented for INT-2 convolutions, by featuring dedicated *packing* and *unpacking* functions.

In the context of a mixed-precision convolution kernel, the precision of the *ifmaps* determines the specific *im2col* function to be used, the precision of the *weights* determines the specific *MatMul* kernel, while the *ofmap* determines the specific *QntPack* kernel.

[2]The thresholding approach can be replace, with the same computational cost but lower memory footprint needed to store the threshold values in the memory, by a quantization step consisting of a MAC operation, one shift and one clip instruction per each accumulator to be quantized back into the desired precision

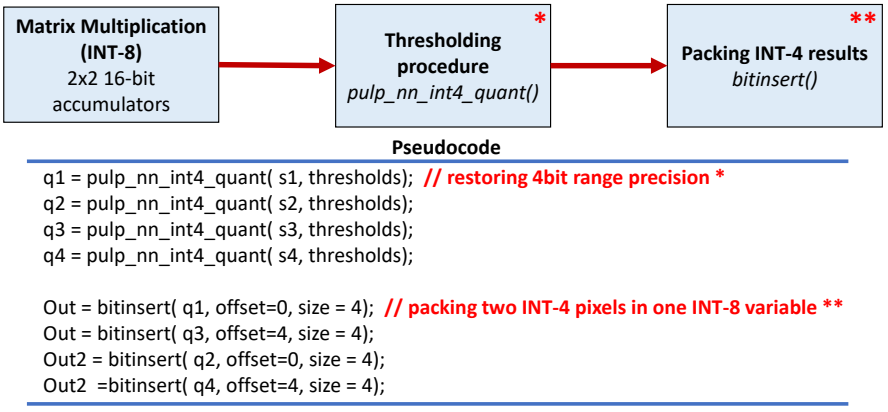


Figure 3.5: The compression procedure for INT-4 data types.

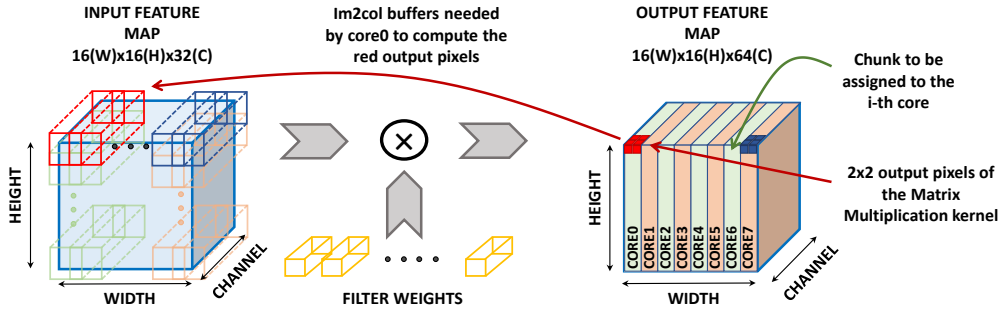


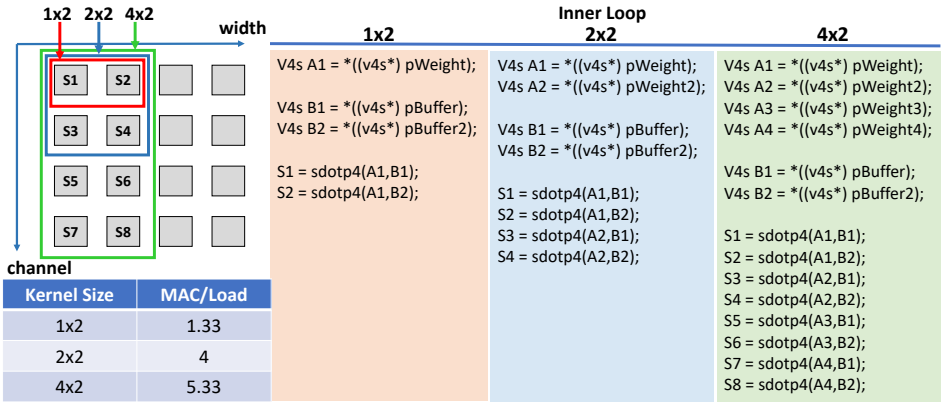
Figure 3.6: The right side of the figure shows how the chunks are assigned to the 8 cores of the PULP cluster. To take advantage of the HWC data-layout each chunk is built along the spatial dimension of the output feature map. The left side gives a graphical intuition of the need each core has to create its private im2col buffer. Considering the 2×2 matrix multiplication kernel each core requires two private buffers of such type.

Binary Convolution Kernel

For the INT-1 data representation no casting/unpacking is needed because of the natural support provided by the ISA for binary operations. We exploit the bitwise instructions to implement the convolution kernel, which is based on bitwise XNOR operations between binary weights and binary inputs. The accumulator is filled by counting the number of ones occurring after the XNOR. To this purpose we use *popcnt* built-in. The 16-bit accumulator is compared with a single threshold and results either in a zero or one, stored back into memory by means of the *bitinsert* built-in function.

3.3.2 Multi-Core Execution

As discussed above the *convolution kernel* execution consists of two phases: the im2col function and the matrix multiplication kernel. The proposed data-parallel multi-core optimization is motivated by the HWC format used to store pixels and weights and



Kernel Size	MAC/Load
1x2	1.33
2x2	4
4x2	5.33

Figure 3.7: Inner loop of the matrix multiplication considering different sizes of the kernel.

by the two phases of the dataflow. Because of the HWC format, it is convenient to split the workload along the spatial dimension of the output feature map, in a way that each core computes the full set of M output features for a given output spatial coordinate, as shown in Figure 3.6. To implement this strategy, each core requires a private im2col buffer. More specifically, if we consider the 2×2 kernel, each core must allocate and load two im2col buffers before running the matrix multiplication kernel. Therefore, the parallelization boost comes at the cost of a small amount of additional memory footprint for the extra im2col buffers, which in the worst case (eight cores configuration) is about 9% of the total when considering 16×16×32 sized input feature map, 16×16×64 sized output feature map and 64×3×3×32 sized 3D convolution filter. The weights instead are shared among the cores.

Since the fully connected layer generates a set of neurons as output (i.e., the output feature map does not extend along any spatial dimension), the only dimension along which we can split the workload is the channel. We assign a balanced number of neurons to be computed to each core. The parallelization of the ReLu and the Max Pooling kernel is straight-forward: the chunk to be assigned to each core is a balanced group of pixels along the entire input feature map.

3.3.3 Matrix-Multiplication Structure Optimization

To further increase the throughput of a memory intensive kernel such as matrix multiplication, it is important to reduce the cost of loading the operands into the registers as much as possible, by maximizing the *data reuse* at the register file level.

The direct implementation of the Equation (2.6) would be inefficient since, from a computation perspective, two loads are required (one to fetch an im2col element and one to fetch a weight parameter) to feed the MAC instruction. In this scenario, one load

stall will be necessarily paid, degrading the IPC metric and reducing the throughput. To avoid the stall, multiple output data can be computed within the inner loop of the dot product routine, i.e., the inner loop of the matrix multiplication kernel.

When applying equation (2.6) to compute the output data at the spatial coordinate $(x+1, y)$, the formula becomes:

$$O(m, x+1, y) = \text{dot}\Big(W(m),\ im2col(x+1, y)\Big)\ . \tag{3.1}$$

We can notice that the same subset of weights is used in the computation of the output data at coordinates (x, y) and $(x+1, y)$. What changes is only the im2col buffer. When operating on these two point simultaneously, the inner loop consists of two dot product operations, which are performed over two different accumulators. By reusing the register that stores the elements of $W(m)$ along the spatial dimension we can set two *sdotp4* operations at the cost of one additional load (three in total), needed to fetch the elements of the second im2col buffer. So doing, we build the 1x2 sized kernel and increment the MAC to load ratio. If extending this strategy also to the feature dimension, the inner loop of the convolution can operate on a 2×2 sized kernel, i.e. computing four accumulations related to two features of two separate output pixels (x, y) and $(x+1, y)$. Such a kernel size is the one used by ARM CMSIS-NN. In this case, an additional subset of weights, $W(m+1)$ is needed and, at the cost of four loads, we can perform four *sdotp4* operations in the inner loop. By means of this upgrading, the MAC to Load ratio grows up to 4.

Let us consider the 4x2 sized kernel, which means we want to compute two adjacent spatial pixels along four consecutive channels of the output feature map. Following what we said before, we need to build two im2col buffers, and we need four different subsets of weights. The elements loaded in the register file are reused similarly as presented before to maximize the MAC to Load ratio. Figure 3.7 explains the concept of register file data reuse. As a counterpart, we can explore the 2x4 sized kernel. In this case, the reasoning is reversed. The MAC to load ratio we can achieve in both cases is 5.33, as we compute 32 MACs at the cost of 6 load operations, in a single run of the inner loop.Thus we expect a better throughput with respect to the 2×2 sized area. It is interesting to notice that in the 2x4 case, the memory footprint is slightly higher than the 4x2 sized kernel because of the two additional im2col buffers. For the same performance, the former is thus to be preferred between the two.

It is important to notice that the upscaling of the kernel size is limited by the resources available in the register file to store operands and accumulators, thus limiting the *data reuse* design space at this level. We explore such a space to find the best

register file data reuse condition which maximizes the throughput. The experimental results and further considerations are provided in Section 3.4.3.

3.4 Results and Discussion

The solutions presented in this chapter are evaluated on the off-the-shelf GAP8 [20] microcontroller, which is an embodiment of the target PULP architecture discussed in Section 2.2. In this section, the experimental results and the related discussion are reported.

3.4.1 Comparison with RV32IMC ISA

To evaluate the proposed library, which exploits the DSP extensions available on the RI5CY processor [25], we first compare the optimized single core execution of the convolution kernels with respect to a corresponding *RV32IMC* ISA implementation, sweeping all the INT-Q datatypes supported. This evaluation is performed by benchmarking a convolution kernel operating on a 16x16x32 input tensor (HWC data-layout) with a filter size of 64x3x3x32 (CxkwxkhxM). We consider the convolution kernel as its workload is dominant when inferring an entire QNN (about 96 % on the CIFAR-10). As a second term of comparison, we run the kernels on off-the-shelf STM32H743 [31] and STM32L476 [30] commercial microcontrollers based on ARM CORTEX-M7 and CORTEX-M4 cores respectively, using the CMSIS-NN [22] library. To run the sub-byte quantized version of the convolution layer on such MCUs, we refer to [23]; the extension to the CMSIS-NN library is open access[3]. The results of the comparison are presented in terms of speedup with respect to the *RV32IMC* implementation and reported in Figure 3.8.

We achieve the best speedup on the INT-8 convolution kernel, mainly thanks to the 8-bit SIMD *sdotp* instructions. The ARM ISA features support for 16-bit instructions only, dividing by a factor of 2 the MAC throughput with respect to the RI5CY processor. Moreover additional rotate instructions are required on ARM architectures to pack 16-bit vector data to feed the MAC units [23]. Finally, hardware loops provide another factor of improvement with respect to ARM. Thanks to these extensions we outperform by 2.54× and 4.51× the STM32H7 and L4 MCUs respectively, despite the CORTEX-M7 processor available in the STM32H7 featuring a dual-issue pipeline.

When considering sub-byte data types, we notice a degradation of the speedup with respect to RV32IMC which passes from 8.8× (INT-8) to 3.69× and 4.22× for INT-4 and

[3] https://github.com/EEESlab/CMSIS_NN-INTQ

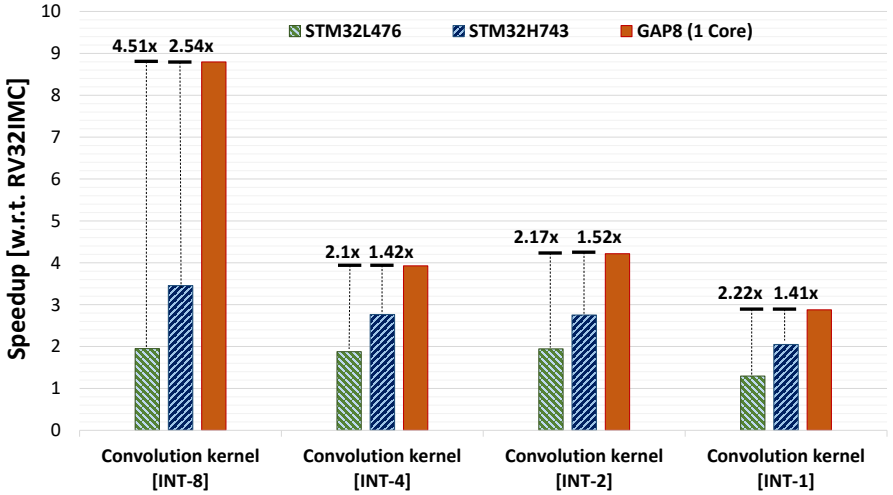


Figure 3.8: Speed-up of PULP-NN conv kernels (single core execution on GAP-8) and CMSIS-NN conv kernels (on STM32H7 and STM32L4) with respect to RV32IMC ISA.

INT-2 data respectively. Such degradation is due to the additional instructions to unpack and cast INT-2/4 operands to INT-8 ones. Although these operations are implemented with *bextract* and *pack4* instructions, they do not achieve the same speedup as the INT-8 convolution kernel, limiting the overall speedup for sub-byte kernels, still leading to a speedup of 1.42× and 2.1× with respect to STM32H7 and STM32L4 for INT-4 kernel, respectively, and a speedup of 1.52× and 2.17× with respect to H7 and L4 for INT-2 kernel, respectively. The ARM CORTEX-M7/M4 processors do not have ISA support for efficient bit manipulation instructions nor for popcount instruction which is helpful for the INT-1 case. However most of the computational load of this kernel is implemented with xnor instructions available in all considered ISAs. Hence, the proposed implementation, runs 1.41× and 2.22× faster than the extended CMSIS-NN solution on STM32H7 and STM32L4 respectively.

3.4.2 Multi-Core Execution Results

This section focuses on the analysis of the multicore optimization of the kernels. Figure 3.9 shows a comparison of the convolution kernels running on the 8-core cluster of GAP-8 with respect to the equivalent CMSIS-NN implementation on STM32H7 and STM32L4. It is possible to notice that, due to the additional operations required to execute sub-byte kernels, their overall cycles/MAC are 0.186 for INT-4 and 0.181 for INT-2, both 2.4× higher than the INT-8 case. However, we can notice how the software-efficient exploitation of the parallel processors cluster provides almost linear speedups (7.16× to 7.7×) with respect to the single core configuration, leading to a dramatic improvement of performance with respect to the equivalent execution on sequential

Configuration	Nr. insns	I$ stall cycles	TCDM cont. cycles	Load stall cycles	Total exec. cycles	Speedup
Convolution						
1 CORE	2546k	1.3k (0.05%)	0	18k (0.7%)	2586k	1×
2 CORES	1286k	4.5k (0.35%)	1.4k (0.11%)	11k (0.85%)	1299k	**1.99**×
4 CORES	636k	5.7k (0.86%)	3.8k (0.56%)	5.5k (0.83%)	660k	**3.92**×
8 CORES	318k	21.5k (5.96%)	6.6k (1.83%)	2.7k (0.75%)	361k	**7.16**×
Fully connected						
1 CORE	20.7k	0.03k (0.09%)	0	0	33k	1×
2 CORES	10.4k	1.1k (6.25%)	1k (5.69%)	0	17.6k	**1.89**×
4 CORES	5.2k	0.1k (1.19%)	0.2k (2.38%)	0	8.4k	**3.92**×
8 CORES	2.6k	0.1k (2.27%)	0.3k (6.81%)	0	4.4k	**7.52**×

Table 3.2: The table shows the multicore execution profiling of the kernels. The measurements for multicore configurations are reported as an average of the measurements taken on each core. The percentage value highlights the impact of each measured contribution on the total execution cycles.

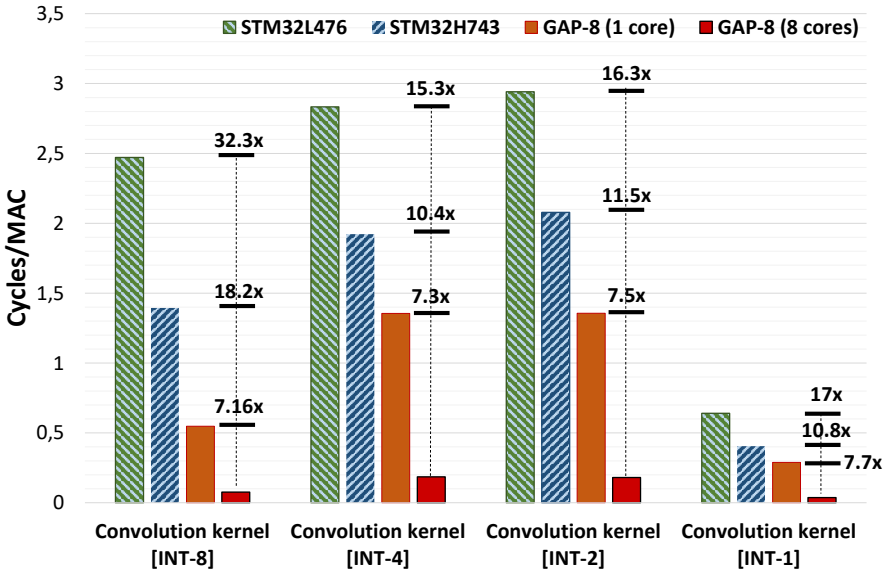


Figure 3.9: Comparison in terms of cycles/MAC between the PULP-NN conv kernels on one/eight core(s) of GAP-8 cluster and CMSIS-NN conv kernels on STM32L4 and STM32H7.

RV32IMC (where the overall speedup passes from 8.8× of the single-core execution to up 63× when considering 8-cores) and on single-core ARM architectures (10× to 32×). This huge performance gain enables the exploitation of the benefits of heavily quantized neural networks in terms of memory footprint, still performing one order of magnitude better than state-of-the-art ARM-based implementations.

To provide more insight on the multi-core optimizations, an exhaustive study of the performance achieved on the parallel cluster of GAP-8 is presented. First, the measurements of the amount of executed instructions per each core providing an indication of the Amdhal's limit of the kernels are reported, i.e. the amount of cycles lost due to

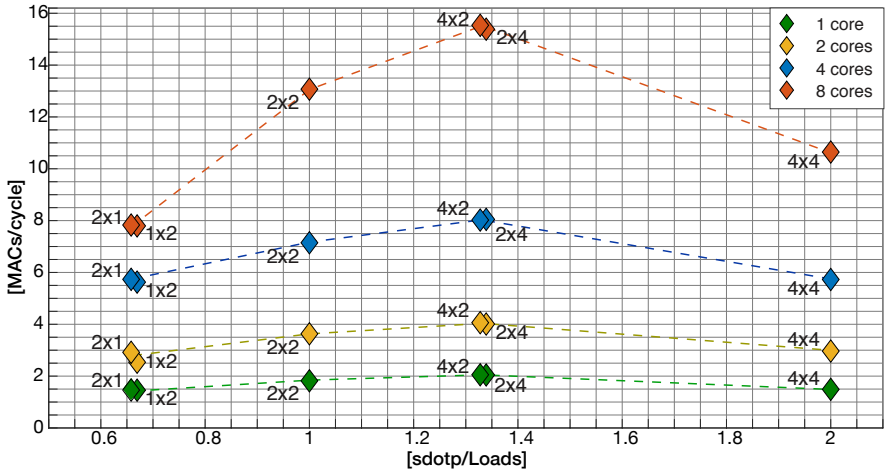


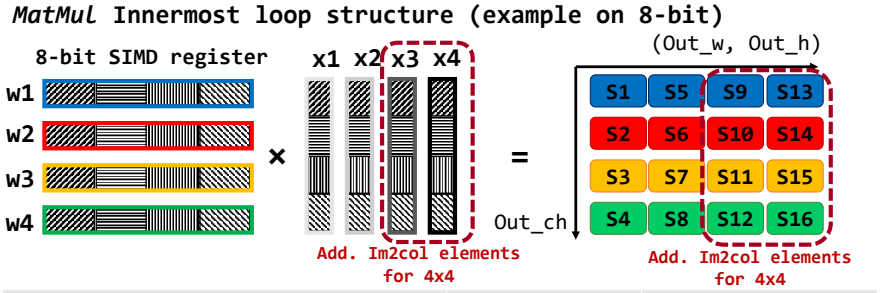
Figure 3.10: Performance of the convolution layer considering different sized matrix multiplication kernels. On the x-axis we show the sdotp to load ratio to clarify how many *sdotp4* (equivalent to 4 MAC) we can set with one load. The label of each point of the graph, in the form of $a \times b$, specifies the kernel size considered. a is the number of output features computed by the kernel, b is the number of output activations.

non-parallelizable code. As a second point, the number of cycles in which the cores are not waiting on a barrier (active cycle) are presented and, in the end, the architectural sources of overhead: number of cycles lost due to contention on the shared TCDM, cycles lost due to instruction cache stalls and cycle lost due to load stalls (read after write). The results for the convolution and fully-connected kernels are summarized in Table 3.2.

Considering the convolution kernel, a Speedup of 7.16× with eight cores is achieved. By analyzing the table one can notice that the Amdahl's limit of the kernels is around 8× (thus, ideal), but a small number of cycles due to architectural overheads is lost: the 67% of this overhead is due to I$ non-idealities, 8% is due to load stalls and 20% is due to TCDM contention, which is reasonable as there are eight cores that access the same shared L1 memory. The number of I$ stalls increases with the number of cores due to the increasing contentions in the shared cache banks [79] (the banking factor of 8 can not completely remove the conflicts), on top of the I$ misses due to the large inner loop of the kernel. The parallel execution of the fully connected layer presents a speedup higher than the convolution kernel mainly thanks to the reduction of I$ stalls due to the smaller size of the kernel. The speedup is never lower than 7× also when considering the max-pooling and ReLU kernels running on eight cores.

3.4.3 Kernel Exploration

The exploration of the matrix multiplication kernel size design space is carried out for the INT-8 operands, considering sizes ranging from 1×2 to 4×4. The results are



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)

Figure 3.11: Layout and hardware resources (registers) of the "4×2" and the "4×4" layouts of the *MatMul* kernels. 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 efficiently the RI5CY register file.

summarized in Figure 3.10. A peak throughput of 15.5 MACs/cycle is reached when we consider a convolution kernel with a 4×2 sized matrix multiplication kernel running over eight cores of the cluster, achieving a result of just 1.01 LD/ST per MAC. This result translates in an overall efficiency of 49% in terms of MAC utilization, only a factor of 2 from the theoretical peak achievable (32 MACs/cycle) on a cluster of eight programmable cores with SIMD MAC units, i.e. considering the MAC units constantly fed. Nearly the same throughput is achieved with the 2×4 sized kernel, as the almost overlapping points in the graph suggests. Then, the optimal sized kernel has been chosen taking into account also the extra memory footprint needed to build the im2col buffers in the two configurations, which results to be lower for the 4×2 solution (see section 4.3.3.3 for more details). As regards the 1×2, 2×1 cases, they appear to be inefficient, as the amount of data reuse is meager and we pay the overhead due to the higher number of loads. For these configurations, the MAC to load ratio is slightly higher than 1. The 4×4 case instead would demonstrate to be the best, since the first indication of ideal data reuse is equal to 8 (MAC/load). However, to set a 4×4 sized matrix multiplication kernel inner loop we should have at least 24 registers available (16 for the accumulators and 8 for the operands), while the target RISC-V, like most MCU-dedicated micro-architectures, has a register file with 32 general purpose registers. With only eight usable registers, the compiler has to spill variables to the stack to make room for the accumulators and operands, leading to significant performance degradation. This reasoning is summarized in Fig. 3.11.

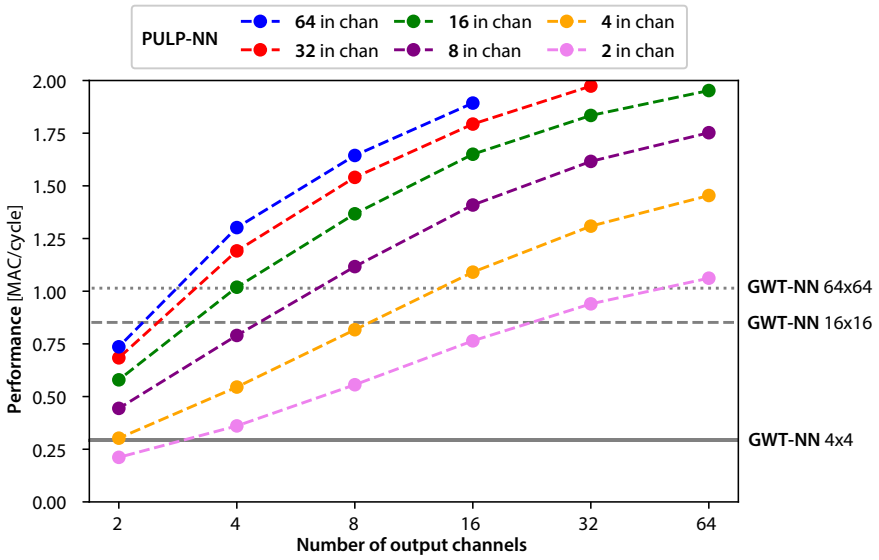


Figure 3.12: Comparison between PULP-NN using a 4×2 kernel and the best result obtained by GWT-NN.

3.4.4 Comparison with GAP8 Native Library

The library presented in this chapter is also compared with the optimized multi-core kernels that are openly distributed by GreenWaves Technologies as part of a proprietary tiling solution[4] and tailored for the GAP8 processor, called GWT-NN in this dissertation. This section compares the performance of PULP-NN on INT-8 data with that provided by GWT-NN, focusing on a 3×3 kernel in terms of filter size as a representative example constituting the bulk of most SoA DNNs.

Differently from PULP-NN, GWT-NN operates spatially on Channel Height Width (CHW)-formatted data with explicit convolution filters working in a sliding window fashion, and accumulation over an appropriately sized INT-32 buffer. In the innermost loop, the GWT 3x3 kernel uses the register file to implement a sliding window and uses three *sdotp4* instructions to implement a total of 9 multiply-accumulate operations. [25] and [7] report further details with respect to this convolution kernel.

Figure 3.12 shows a comparison between the two libraries when running on a single core of the GAP-8 cluster, in terms of performance in MAC/cycle. For PULP-NN, the performance is swept by changing the number of input and output channels between 2 and 64 (only results from configurations fitting the L1 are shown). We chose the biggest input spatial size (24x24) for which configurations with 64 input or output channels fit L1. Conversely, for GWT-NN, performance is substantially independent of the number of in/out channels, but only on the spatial size of the input image; therefore, their

[4] https://github.com/greenwaves-technologies/autotiler.

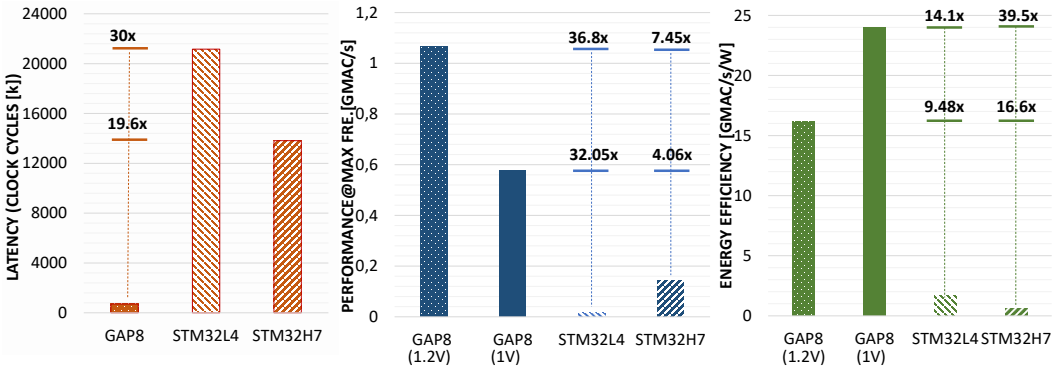


Figure 3.13: This figure shows the execution cycles, the performance (at the maximum frequency) and energy efficiency (at the lowest consumption configuration) to infer the entire QNN on GAP8, STM32L4 and STM32H7 microcontrollers.

input/output channels are fixed at 4 and their input size is swept between 4, 16, and 64 pixels height/width.

As visible from Figure 3.12, PULP-NN outperforms GWT-NN for all small images, and in most cases of spatially bigger images by a significant margin. This is due to a combination of two effects: the 3x3 sliding window requires three loads and three *sdotp4* per output pixel, yielding a lower *sdotp4* per load ratio (1) with respect to the 4×2 PULP-NN kernel (1.4); moreover, only three MAC are used per each *sdotp4*, yielding a further loss of 25% in terms of efficiency. Consequently, the GWT-NN kernel is mostly competitive when the spatial size of the feature maps is much higher than the number of channels, e.g., in the first layer of a CNN. While, when the number of input/output channels is high, which typically represents the majority of the workload for state-of-the-art deep networks topologies [80], PULP-NN can achieve as much as a +89% speedup with respect to GWT-NN.

3.4.5 Comparison with the *State-of-the-Art*

To assess the library performance on an inference task, we run a full QNN, trained on CIFAR-10 dataset, on GAP-8, using PULP-NN back-end library. For comparison purposes, we run the same network also on State-of-the-Art edge of IoT ARM Cortex-M based microcontrollers (STM32H7 and STM32L4), using CMSIS-NN. STM32H7 and STM32L4 were chosen as representative of popular high-end and low-end MCU systems, with a clear trade-off between performance and energy efficiency. The comparison with these two popular computing platforms allows to analyze where our results lay in terms of trade-off between computing performance and energy efficiency. The implemented network topology is composed by three convolution layers and one fully-connected layer,

consisting of 26.7 k parameters and 6.56 MMACs in total[5]. The weights and the activations are quantized to INT-8 format. Such a topology is already used on IoT edge devices (MCUs) and also used by ARM to validate Neural Networks on low-power microcontrollers such as STM32L4 or STM32H7.

On GAP-8, the RGB image is initially stored in the L2 memory and brought in the L1 memory before the start of the inference task, through a DMA transfer. The activation values are then kept in the L1 memory to save on memory transfer overhead. Before the execution of each convolution or linear kernel the weights, initially residing on L2 memory, are brought in L1 through DMA as well. Also the im2col buffers are kept in L1 memory. On the STM32L4 microcontroller, the entire network is stored in the first level of memory, which consists of 128 kB SRAM. On STM32H7 the network is stored in SRAM as well and we enable also the harware data cache which is provided by the MCU architecture.

In the single core configuration, we are able to infer the entire network in 28.6 ms, when GAP-8 runs at 170 MHz. We achieve almost linear speedup when considering two and four cores, 1.99× and 3.79× respectively. With eight cores the speedup is slightly less than 7×. Figure 3.13 shows the comparison of PULP-NN implementation of the network on GAP-8 with respect to the CMSIS-NN implementation on STM32H743 and STM32L467 in terms of execution cycles, performance (i.e. also considering the maximum operating frequency of the devices), and energy efficiency.

Our PULP-NN CIFAR-10 achieves a peak performance of 1.07 GMAC/s at the frequency of 170 MHz and the supply voltage of 1.2 V on GAP-8, inferring 241 frame per second (fps) with an energy per inference of 0.27 mJ/frame. The performance is 7.45× better than the STM32H7 and 36.8× better than the STM32L4. The energy efficiency achieved at this operating point is 16.1 GMAC/s/W, 16.6× higher than the STM32H7 and 9.48× higher than STM32L4. At the same time, at the best energy point, at the supply voltage of 1V, PULP-NN achieves a performance of 577 MMAC/s on GAP-8, with energy efficiency of 24 GMAC/s/W, inferring 127 fps with 0.19 mJ/frame, and outperforming STM32H7 by 4.06× and STM32L4 by 32.05× in terms of performance and by 39.5× and 14.1× the same devices respectively, in terms of energy efficiency.

[5]The layer parameters can be found at: https://github.com/ARM-software/ML-examples/tree/master/cmsisnn-cifar10

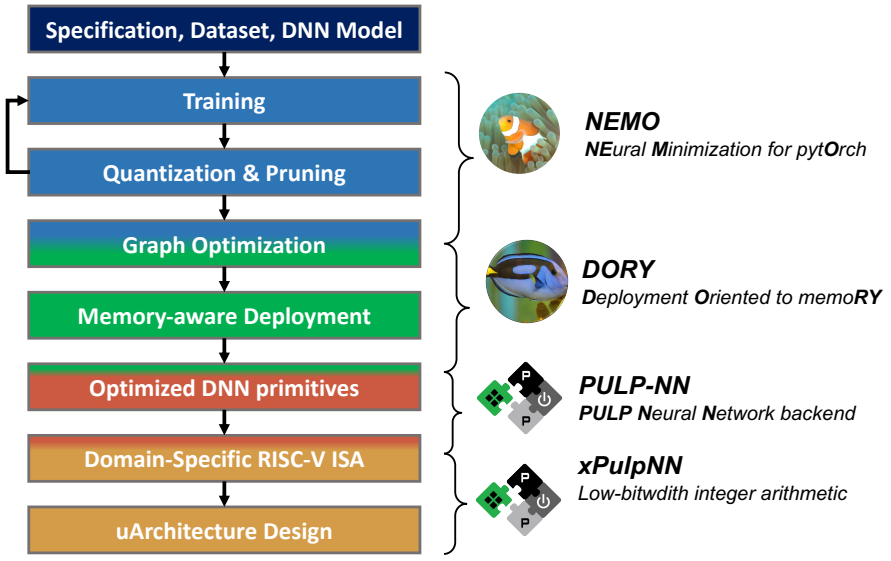


Figure 3.14: Vertical integration flow for the end-to-end deployment of Neural Network models on top of PULP-based systems. The framework includes *NEMO*, which from PyTorch NN model generates its integer representation deployable to the target system: *DORY*, which efficiently manages the optimal memory management of the sytem; *PULP-NN*, which features the optimized DNN primitives and *xPULPNN*, a custom set of RISC-V ISA extensions to accelerate DNN routines, discussed in Chapter 4

3.4.6 End-to-End inference of Neural Networks on PULP

The contributions presented in this chapter flow into an open-source library, namely PULP-NN [6]. The library optimizes the heavy computation of inference tasks assuming the activations and the weights are already stored in the closest level of the memory hierarchy of the system, but it does not consider the orthogonal problem of optimizing the data movements across the different memories. This aspect of the inference is essential to consider real-world sized neural network models which might not fit the usually small (few KiloBytes) L1 scratchpad memory of the MCUs. Therefore, PULP-NN is integrated as back-end library into a vertical deployment flow which includes a tool for efficient and automatic memory management during the execution of the inference tasks, namely *Dory* [38], and a tool for quantizing and deploying a high-level Neural Network model (e.g. from PyTorch) into a integer-represented quantized network ONNX graph, namely *NEMO* [82]. The vertical end-to-end framework is shown in Figure 3.14.

DORY (Deployment Oriented to memoRY) is an automatic tool to deploy DNNs on low cost MCU architecture like PULP, which replace caches with scratchpad memories (typically with less than 1MB of on-chip SRAM memory) to reduce area overheads and increase energy efficiency. DORY manages multi-level memory tiling aiming at the deployment of realistically sized DNNs on memory-starved MCUs. Relying on Constraint

[6] https://github.com/pulp-platform/pulp-nn;
https://github.com/pulp-platform/pulp-nn-mixed

Programming (CP) optimization, the tool matches on- and off-chip memory hierarchy constraints with DNN geometri- cal requirements, such as the relationships between input, weight, and output tensor dimensions. Through a set of defined heuristics, the performance of the CP solution can be optimized specifying the target hardware platform and the back-end library, the PULP platform with three-levels of memory and the PULP-NN library, in the case-study considered. The third block that composes DORY is a code generator that uses tiling solutions to produce ANSI C code for the target platform, with all data L3-L2-L1 orchestration implemented as a fully pipelined, triple-buffered DMA transfers and integrated calls to the computational back-end (PULP-NN). More details on the structure of DORY can be found in [38].

DORY with the PULP-NN back-end have been tested on the deployment of full-networks that are already used as bench-marks for many edge-oriented works [80]. All the networks were run on GWT GAP-8, verifying all intermediate results as well as the final result of end-to-end runs against a PyTorch-based bit-accurate golden model for QNNs [82], to confirm the correct functionality of the DORY framework and the PULP-NN backend.

Table 3.3 showcases a full comparison in terms of energy efficiency (GMAC/s/W), throughput (GMAC/s), latency, and energy per frame. Different variations of the MobileNet-v1 have been compared, with the same topology but a different number of channels or input dimensions. For state-of-the-art, the biggest networks that fit the on-chip/off-chip memory of the STM32H7 [31] and GAP8 [20], respectively (compatible with the ones deployed with DORY), are shown. As can be noticed from the Table, DORY on MobileNet-v1 achieves up to 13.19× higher throughput in MAC/cycles than the execution on an STM32H7 (on 0.5-M.V1-192), using the best framework (X-CUBE-AI [83]) currently available. On different operating points, a 7.1× speed-up (1.78 vs. 0.25 GMAC/s) is achieved and 12.6× better energy efficiency, given the different frequencies and power consumption of the two platforms. Compared with GWT-proprietary and partially closed-source AutoTiler run on the same GAP-8 platform, the results presented show that DORY performs on average 20.5% better. The advantage lies in *1)* the more efficient backend (PULP-NN) and *2)* the heuristics, which guarantee that the tiling solution is optimized for the PULP-NN execution model.

Fig. 3.15 depicts the power profile of the end-to-end execution of a MobileNet-v1 (1.0 width multiplier, 128 × 128 resolution) on GAP-8, with both the cluster and the fabric controller running at 100 MHz. The power consumption of the cluster domain (including 8 RI5CY cores, the L1 and the Cluster DMA) and of the I/O domain (including 1 RI5CY core, the L2, and the I/O DMA) is shown separately in two separate subplots. In the cluster domain, power is dominated by the cores when the computation is in the active

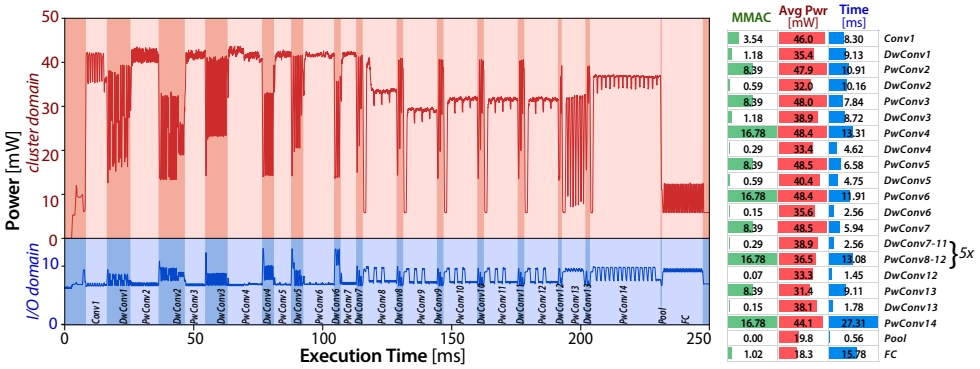


Figure 3.15: In the left part, the 1.0-MobileNet-128 power profile when running on GAP-8 @ $f_{\text{cluster}} = f_{\text{io}} = 100\,\text{MHz}$ and $V_{DD} = 1\,\text{V}$. On the right, number of MAC operations, average power, and time for each layer of the network. Power was sampled at 64 KHz and then filtered with a moving average of 300 µs.

phase. Small valleys within a layer are given by (short) waits for the end of a memory transfer where the cores are all idle, or by Cluster DMA calls where a single core is active. In the I/O domain, we can notice the I/O DMA consumption spikes: at the beginning of each layer, the weights of the following one are transferred from L3 to L2.

3.4.7 Discussion

The solutions proposed in this chapter demonstrate that coupling optimized software libraries with a parallel ultra-low power computing platform it is possible to achieve energy proportionality where, as opposed to commercial ARM-based solutions, the performance must not be traded with the energy efficiency, paving the way to fully software programmable CNN inference at the extreme edge of the IoT. However, sub-byte and mixed-precision kernels still suffer from drop-off in performance when compared to the INT-8 ones, despite their execution on GAP-8 performs more than one order of magnitude better with respect to MCU-based SoA solutions. The overhead, as highlighted in Section 3.4.1, is due to the hardware support of the target architecture only for 8-bit SIMD instructions, which makes necessary to introduce additional packing and unpacking functions. The sub-byte and mixed-precision precision QNNs though, provide several advantages when deployed at the edge, since their memory footprint decreases linearly with the bit-width used to represent weights and activations [13], making them more suitable to fit the limited memory capacity of MCU-like devices. Moreover, it has the potential to increase the energy efficiency, crucial for battery-powered devices [43]. Recent research demonstrated that, by exploiting specific retraining techniques, the accuracy drop can be kept under control, leading to a cumulative loss which is acceptable for many IoT applications [16]. Hence the research community is focusing more and more on the study and implementation of strongly quantized NNs. It is therefore important going

further in the work presented in this chapter to exploit fully the potential of heavily quantized networks on fully programmable edge devices. From the hardware perspective, providing the target ISA with sub-byte hardware SIMD operations will be a step forward to eliminate the software overhead and to double, at least, the performance and the energy efficiency with respect to the current optimal 8-bit solution. This topic will be exhaustively discussed in Chapter 4.

Table 3.3: End-to-end execution of image recognition MobileNet-v1 and MobileNet-v2 on GAP8 and STM32H7 MCUs.

Configuration	**Params**	**Work** MAC	**Cycles**	**Perf** MAC/cyc	**Eff** GMAC/s/W	**Perf** GMAC/s	**Lat** lat. [ms]	**Energy** E [mJ]	**Eff** GMAC/s/W	**Perf** GMAC/s	**Lat** lat. [ms]	**Energy** E [mJ]
DORY @ GAP8												
					Low energy 1V @ 100 MHz				Low latency 1.15V @ 260 MHz			
1.0-M.V1-128	4.2 M	186.4 M	23.3 M	8.00	15.68	0.80	233.11	11.89	7.93	2.08	89.66	23.51
0.5-M.V1-192	1.3 M	110.0 M	16.0 M	6.86	13.46	0.69	160.2	8.17	6.82	1.78	61.62	16.16
0.25-M.V1-128	0.5 M	13.5 M	2.8 M	4.74	9.30	0.47	28.50	1.45	4.69	1.23	10.95	2.87
1.0-M.V2-128	3.47 M	100.1 M	19.0 M	5.27	10.33	0.53	190.03	9.69	5.22	1.37	73.09	19.16
GWT AutoTiler @ GAP8												
					Low energy 1V @ 100 MHz				Low latency 1.15V @ 260 MHz			
1.0-M.V1-128	4.2 M	186.4 M	28.1 M	6.64	13.02	0.66	280.80	14.32	6.58	1.73	108.00	28.32
1.0-M.V2-128	3.47 M	100.1 M	19.7 M	5.07	9.95	0.51	197.38	10.07	5.03	1.32	75.92	19.91
X-CUBE-AI @ STM32H7, solutions fitting 2MB ROM + 512 kB R/W RAM @ 480 MHz [81]												
0.25-M.V1-128	0.5 M	13.5 M	26.0 M	0.52	1.07	0.25	51.14	12.67	n.a.	n.a.	n.a.	n.a.
0.5-M.V1-192	1.37 M	109.5 M	212.3 M	0.52	1.06	0.25	442.27	103.49	n.a.	n.a.	n.a.	n.a.

Chapter 4

XpulpNN: QNN Acceleration Through RISC-V ISA Extensions

4.1 Introduction

Resource-demanding QNN workload imposes the new generation of IoT platforms to improve their processing characteristics to meet the latency and energy requirements of AI-enhanced IoT applications, within the power budget typical of the micro-controller class of devices (i.e. few milliWatts). As demonstrated in Chapter 3, lack of hardware supports for sub-byte SIMD operations in the ISA of IoT processors leads to non-negligible performance and computing efficiency degradation, whenever the inference of a sub-byte or mixed-precision QNN is executed. This chapter gives insights on how to solve this issue working at the architectural and micro-architectural level of micro-controller class of devices.

4.1.1 Motivation

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 [13, 15]. 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 QNN workload at the edge.

This trend impacts all the main categories of edge-AI computing platforms introduced already in Section 4.2, i.e. dedicated accelerators, FPGA solutions and embedded Microcontroller (MCU) systems, but with different grades.

Although multi-precision arithmetic units are widely explored in ASIC solutions [49, 55, 56, 68, 70], sub-byte integer arithmetic does not find enough room in the new generation of architectural solutions for MCU-based systems, which are the only candidates to be employed as IoT end-nodes due to their low-cost, low-power and flexible software programmability characteristics.

Coupling programmable cores with specialized accelerators that explicitly deal with sub-byte arithmetic, whithin the MCU architecture, is a solution to enhance the performance of IoT devices, since computational-heavy tasks can be off-loaded to the specialized hardware [20, 26, 84, 85]. Nevertheless, this solution presents the same pitfall described for the ASICs accelerator in Section 3.2: the accelerators work well in the shape and size of networks for which they were designed, but continuous evolution of the network topology poses a severe challenge in the full utilization of the functional units presented in the accelerators.

A most effective approach, that would guarantee the highest flexibility and adaptability to a wide range of DNN models and workloads, consists of extending the ISA of IoT processors with domain-specific instructions to deal with custom workloads without impacting their general-purpose characteristic. However, 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 [25], ARMv8.1-M [26]) data. Hence, sub-byte quantization remains an effective technique to compress the footprint of DNN models on top of these devices [13], but it incurs in performance and energy overhead during the computation, as demonstrated in Chapter 3: 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.

This chapter tackles this problem by presenting the design of 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 presented contributions aim at bridging this ISA and hardware gap to improve the

computing efficiency of heavily-QNN 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.

4.1.2 Contributions

The main contributions are the following:

- The design of 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.
- The integration of the unit into an open-source RISC-V processor [25], 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, the ISA of the core is extended with domain-specific instructions, namely *XpulpNN*. Moreover, the *XpulpNN* extensions are mapped on top of the extended core, and the GCC toolchain is enhanced with machine descriptions of the new instructions to have a full hardware-software interface;
- The integration of the extended core in an eight cores Parallel Ultra-Low-Power (PULP) computing cluster, showing almost linear performance improvements of QNN kernels with respect to the single-core execution;
- The implementation of the PULP cluster integrating the proposed core in GlobalFoundries 22nm Fully Depleted Sylicon on Insulator (FD-SOI) 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;
- The PULP system with the proposed extension is compared with state-of-the-art architectures and software. When running QNN convolution layers, the solution presented in this chapter demonstrates 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.

4.2 Related Work

This section recaps 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 it gives insight on their applicability for the DL deployment at the extreme-edge of the IoT.

4.2.1 Low-Bitwidth Arithmetic in Edge AI Computing Platforms

The efficacy of low bit-width arithmetic architectures for QNNs workload has been widely demonstrated in the domain of dedicated accelerators. For example, in [55] the authors 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 [56], 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 [56] 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 [86]. 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 [20], 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 [84], 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. ARM adopts the same approach coupling the Cortex M-55 [26] with the optional Ethos-55 [85], an accelerator designed to boost machine learning tasks; depending on the configuration, the system can execute 32 to 256 MAC/Cycles.

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 [1] family of microcontrollers, based on the Cortex-M4 cores) and high-end (STM32H7 [2] family embedding the Cortex-M7 cores) micro-controllers supporting DL processing at the edge. On the RISC-V side, the XpulpV2 ISA extensions [25] 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 [20] 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.

Even if the sub-byte integer arithmetic is already adopted in training and quantization flows and ASIC/FPGA-based systems, the ISA of modern MCUs still lack support for low-bitwidth integer arithmetic with lightweight SIMD instructions. The new generation of the ARM ISA for Cortex-M core [26], tailored for the QNN workload, features hardware loops, conditional execution instructions, and 8-bit SIMD instructions like the ones presented in [25]. However, it will not support lower-precision SIMD arithmetic.

The solutions presented in this chapter overcome the limitations described above, outperforming the state-of-the-art hardware and software solutions by at least two orders of magnitude in terms of performance and efficiency and nearing the computing efficiency of ASIC solutions for edge AI.

[1] https://www.st.com/resource/en/datasheet/stm32l476je.pdf
[2] https://www.st.com/resource/en/datasheet/stm32h743bi.pdf

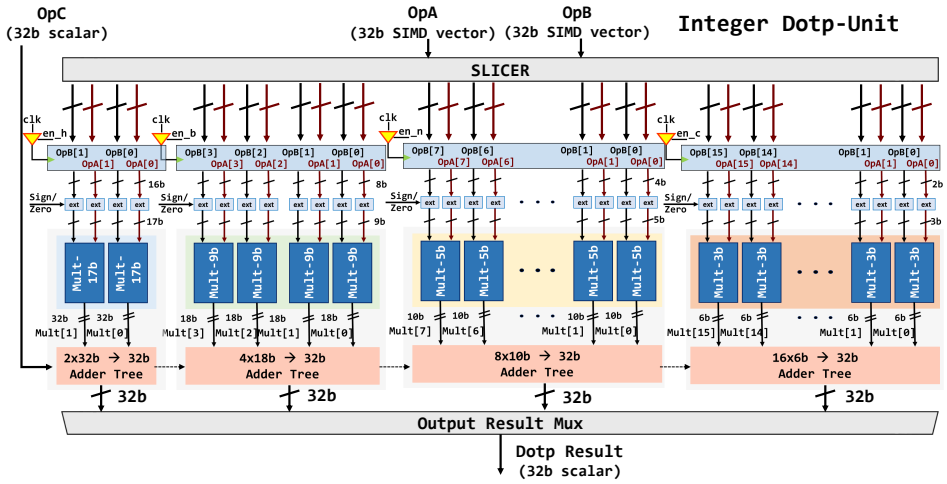


Figure 4.1: Block diagram of the RI5CY Dot-Product Unit. To support the *XpulpNN* SIMD dotp-based operations, the 8×4 and the 16×2 SIMD MAC Units have been added. The figure includes the clock gating blocks needed to reduce the operand switching activity.

4.3 *XpulpNN*

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 [25], 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.

4.3.1 Multi-Precision Dot-Product Unit

The proposed *Multi-Precision Dot-Product* unit, depicted in Figure 4.1, 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 4.1).

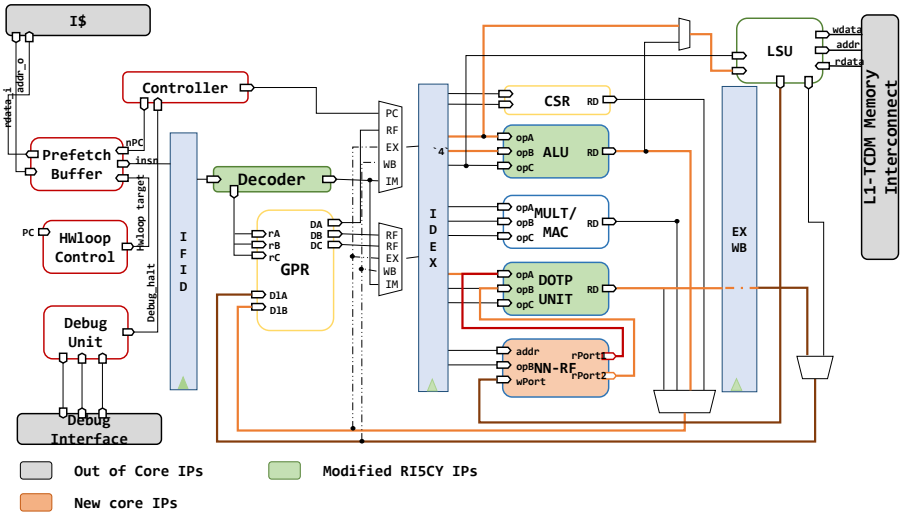


Figure 4.2: The RI5CY pipeline. The Figure highlights the hardware blocks which extend the core micro-architecture to support the *XpulpNN* ISA.

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 4.2. 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 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 4.4.1.

4.3.2 SIMD Instructions 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 4.1, extend the RV32IMCXpulpV2 ISA [25] 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 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.

Table 4.1: 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

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.

4.3.3 Fused Mac-Load operation

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 [28] and already discussed in Section 3.3.3. The innermost loop of the *MatMul* kernel of PULP-NN (Fig. 3.11) 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 × 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 3.11 (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.

4.3.3.1 Compute&Update Instruction

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 4.3 (a) and 4.3 (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 [25]. 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

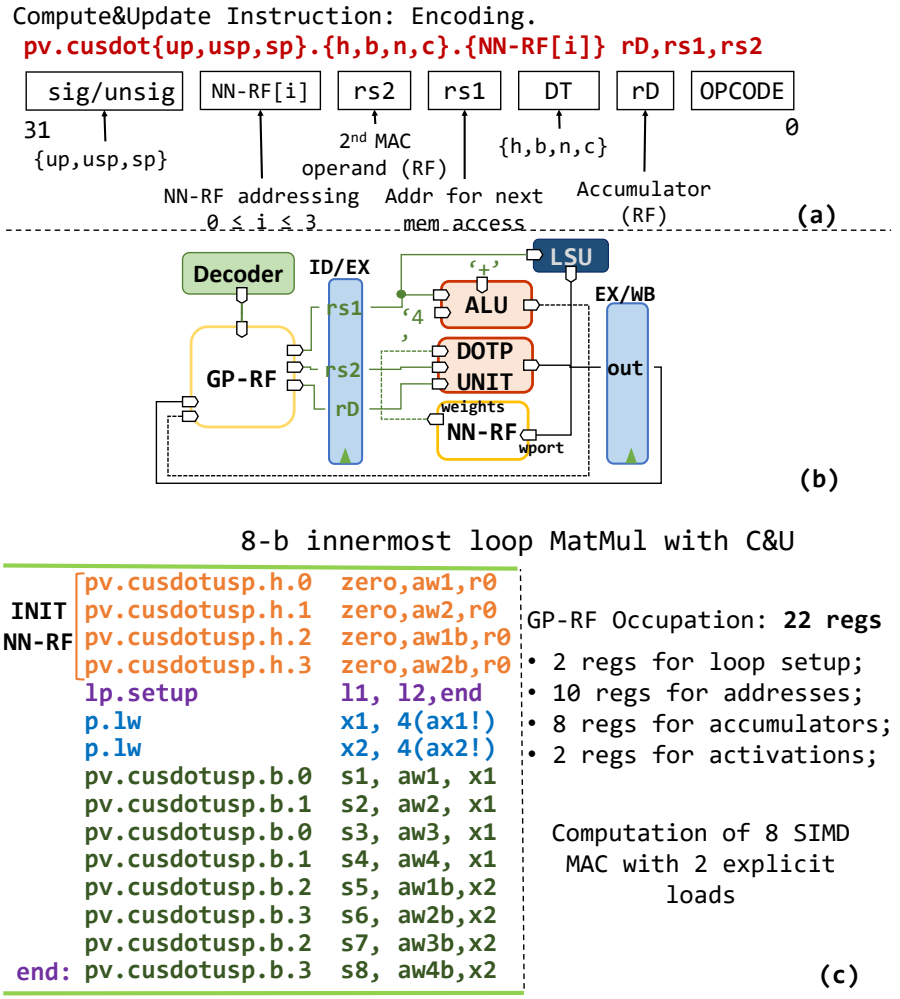


Figure 4.3: In (a), the prototype of the Compute&Update (C&U) instruction is reported: the MSBs encode the interpretation of the operands, "NN-RF[i]" selects the current NN-RF register, "rs1" is the address for the next memory access, "rs2" is the second operand for the MAC unit, while DT encodes the 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.

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 4.3.(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.

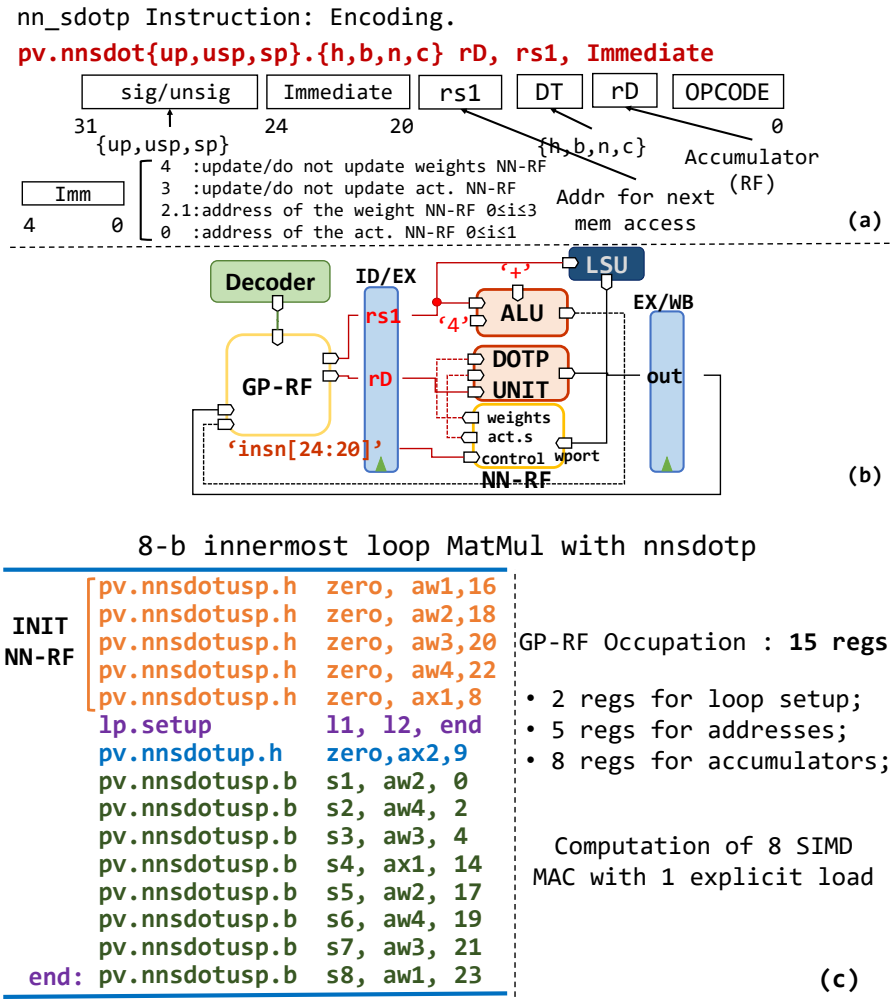
As visible from Figure 4.3.(a), the addressing of the NN-RF registers ("NN-RF[i]" field) is hard-encoded into the instruction to compress as much as possible all the necessary information to execute the C&U in the 32-bits of the encoding space. This causes the addition of four different C&U instructions, each one controlling one register of the NN-RF. 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 4.3.(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 4.3.(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× 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 4.3.(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



8-b innermost loop MatMul with nnsdotp

```
INIT    pv.nnsdotusp.h  zero, aw1,16
NN-RF   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)

Figure 4.4: (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.

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.

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

As visible in Figure 4.4.(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 4.4.(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.

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

```
INIT                pv.nnsdotusp.h  zero, aw1,16
THE                 pv.nnsdotusp.h  zero, aw2,18
NN-RF               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  s16, aw4, 23
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
```

Figure 4.5: 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.

The implementation of the *MatMul* kernel using the *nnsdotp* instructions is reported in Figure 4.4.(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 4.4.(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 "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 4.5. 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.

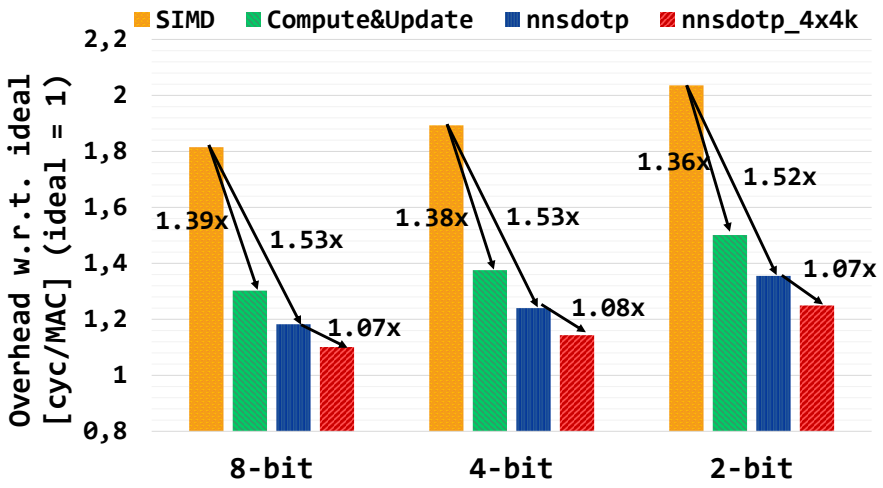


Figure 4.6: Inverse of the Efficiency (lower is better) of the Matrix Multiplication kernel. The bar chart shows the cycles needed to the core to perform one SIMD MAC operation (4x8-bit. 8x4-bit, 16x2-bit respectively). The classical SIMD sdotp (XpulpNN) and the two versions of the MAC&Load instructions (macload, nn_custom) are considered. The nn_custom also allows to enlarge the Matrix Multiplication layout (nn_custom_4x4k).

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 4.6 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 *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.

4.3.4 Integration of the Core into the PULP Cluster

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 [28]. We report in Figure 4.7 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

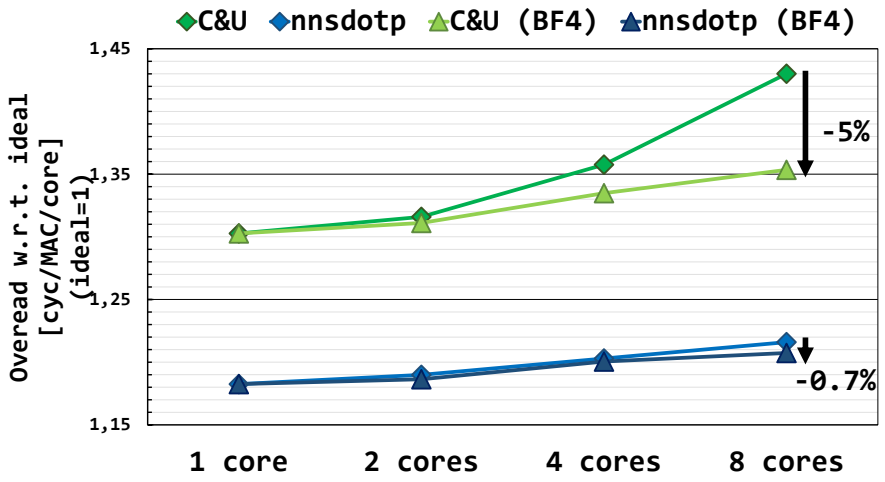


Figure 4.7: Inverse of the MAC Operation Efficiency (lower is better) of the PULP cluster on 8-bit Matrix Multiplication (*MatMul*) kernels.

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 4.7, 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 4.3.3.1, 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.

4.3.4.1 Compiler and 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

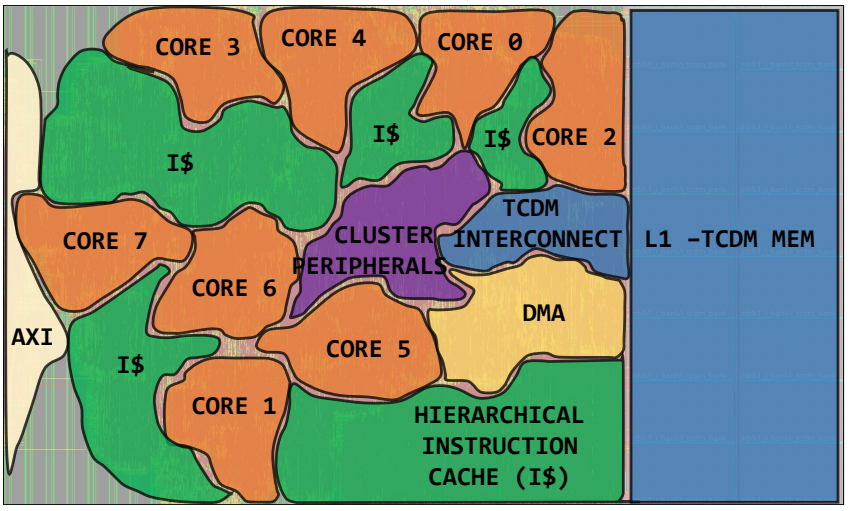


Figure 4.8: Placed and routed design of the PULP cluster with eight extended RISCY cores, supporting the *XpulpNN* ISA.

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 [38], where the backend library is integrated into a vertical QNNs deployment flow.

4.4 Results and Discussion

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 GlobalFoundries 22nm FD-SOI technology.

We synthesize the two clusters with Synopsys Design Compiler-2018.3, and we perform a full place & route flow using Cadence Innovus 17.11, in the worst-case corner (SS, 0.59V, $-40°$/ $125°$). The floorplan of the cluster is reported in Figure 4.8. 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 chapter, 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 [38]. The benchmarked layers operate on a 16 × 16 × 32 input tensor with a filter size of 64 × 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 3.3.1, after the *MatMul* kernel, the intermediate results are compressed back into the desired precision through batch-normalization and activation functions.

4.4.1 Physical Implementation Results

Table 4.2 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 4.2, 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 4.3.2, the duplication of the hardware resources into the Dotp-Unit allows us

Table 4.2: 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%**)

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

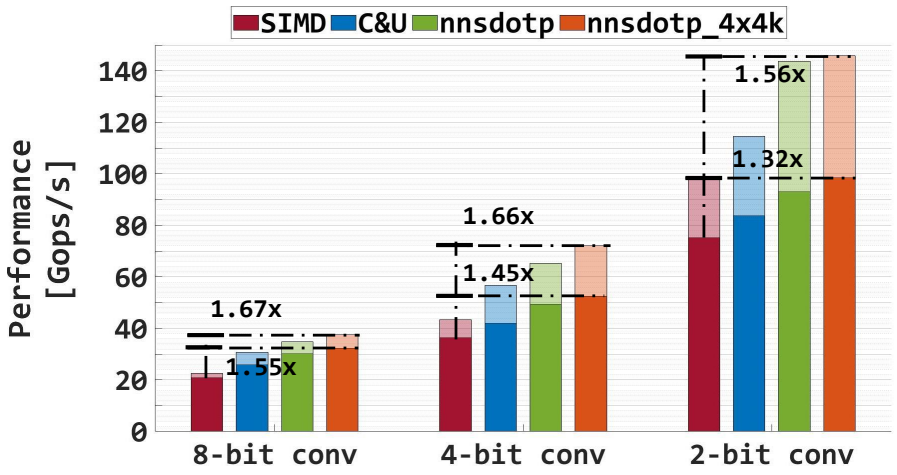


Figure 4.9: Performance of the 8 core PULP clusters over different bit-width precision Convolution kernels, implemented with the instructions presented in this chapter. The lighted bars (higher-performance) refers to the *MatMul* kernel only, while the darker ones include also the quantization procedure (hence, the whole convolution). The cluster runs in the best performance operating point, at 660 MHz, 0.8 V in the typical corner.

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: 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 4.2, 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 chapter, and furthermore showing that we do not jeopardize the energy efficiency of the core on general-purpose benchmarks.

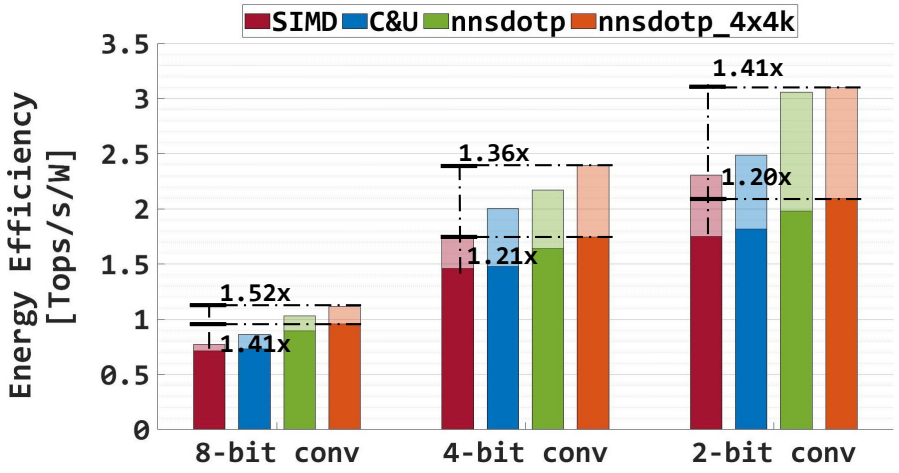


Figure 4.10: Energy efficiency of the convolutions on the 8 core PULP clusters. The graph compares the solutions described in this chapter. The lighted bars (higher-performance) refers to the *MatMul* kernel only, while the darker ones include also the quantization procedure (hence, the whole convolution). The cluster runs in the best efficiency operating point, at 450 MHz, 0.65 V, in the typical corner.

4.4.2 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 4.9 and 4.10 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

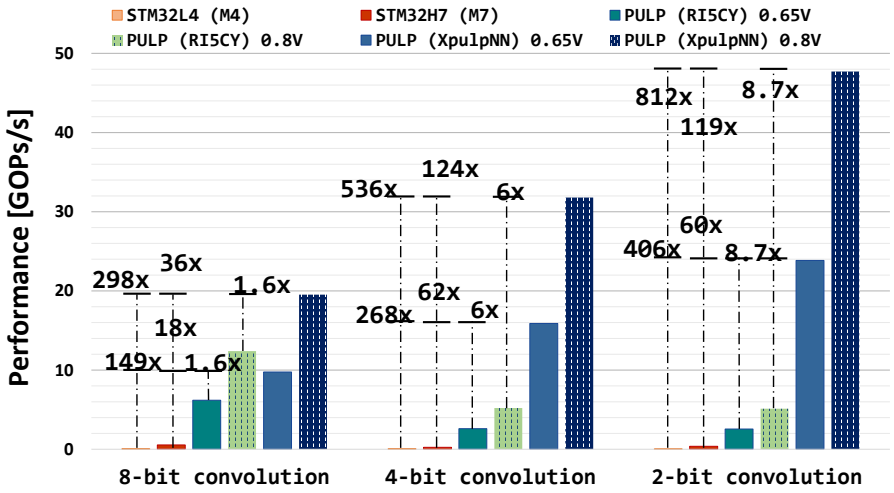


Figure 4.11: The Figure shows the comparison of the solution presented in this chapter with the State-of-the-Art (high-end STM32H7 and low-end STM32L4 MCUs) and with the baseline RI5CY cluter, in terms of performance. The PULP clusters run in two operating points: high-voltage (0.8 V, 400 MHz) and low-voltage (0.65 V, 200 MHz). 8-, 4- and 2-bit simmetric convolution kernels are benchmarked to carry out the comparison.

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 4.9, 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.

4.4.3 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.

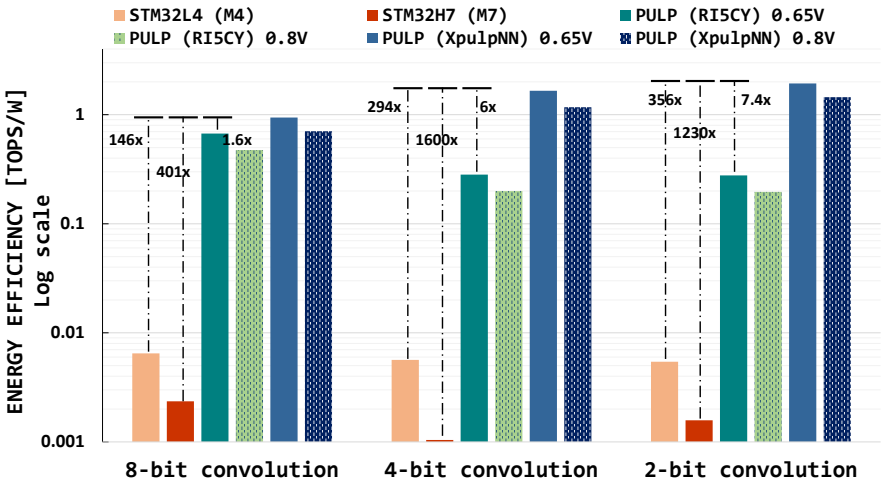


Figure 4.12: Energy efficiency comparison of the solution presented in this chapter with State-of-the-Art and the baseline RI5CY clusters. 8-, 4- and 2-bit simmetric convolution kernels are benchmarked to carry out the comparison.

To carry out the comparison, we run the convolution layers on the RI5CY cluster using the PULP-NN library [28] and on two off-the-shelf STM32H7 and STM32L4 commercial microcontrollers previously introduced in Section 4.2, using the extended CMSIS-NN library [23]. The performance and energy efficiency results are summarized in the Figure 4.11 and 4.12 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 4.9, 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 4.12.

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 FD-SOI technology.

4.5 Silicon Prototype: Dustin

This section presents the silicon demonstration of the concepts and solutions presented in this chapter. Dustin is a test chip taped-out with the TSMC 65nm technology based on the PULP architecture introduced in Section 2.2. It consists of a tiny microcontroller system (called Soc) accelerated by a software configurable MIMD/ SIMD cluster of 16 cores. Dustin aims to demonstrate to be a fully-programmable edge of IoT device capable of enabling efficient parallel execution of computing-intensive kernels, such as convolutions, of modern heavily-QNNs.

Both cluster's and Soc's cores feature RISC-V processors which extend the RI5CY core (introduced in Section 2.3) with mixed-precision SIMD operations in a Dynamic Bit-Scalable Execution context, following the same trail of *XpulpNN*. The novelty of this chip consists also of a software-configurable MIMD/SIMD cluster. When the SIMD mode is active, called *Vector Lockstep Execution Mode (VLEM)*, only one core dispatches instructions, allowing to save energy on extremely regular kernels from an instruction viewpoint (like convolutions and, more in general, matrix multiplications).

4.5.1 Architecture

Fig. 4.13 shows the architecture of DUSTIN. It is built around a tightly-coupled cluster of 16 32-bit RISC-V cores sharing a 128 kB, 32-banks Tightly-Coupled Data

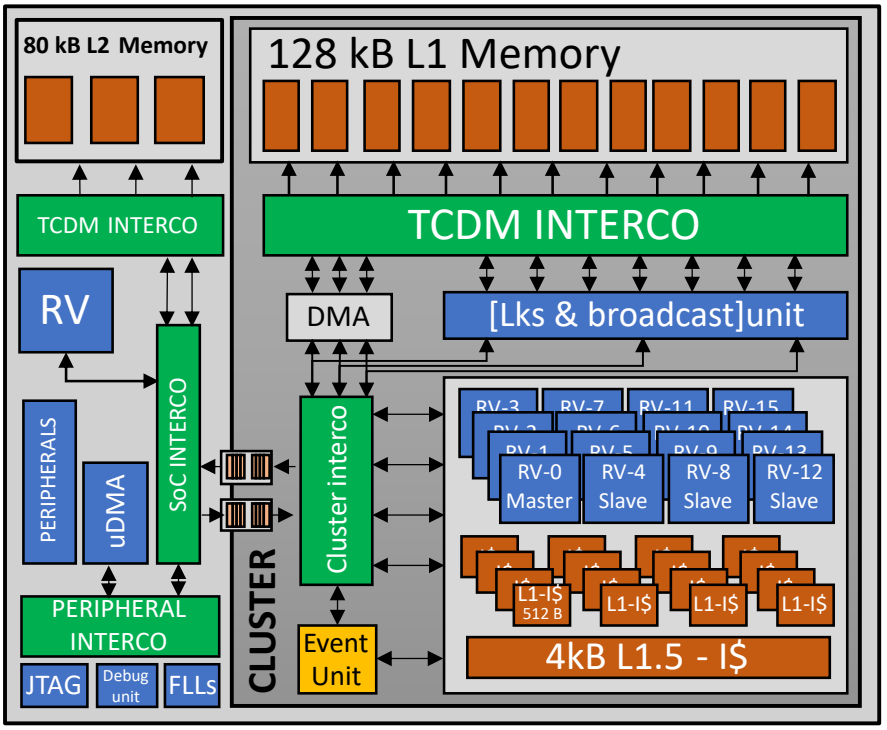


Figure 4.13: Overview of the Dustin SoC Architecture.

Memory (TCDM) through a single-cycle latency logarithmic interconnect (LIC) leveraging a request/grant protocol. The LIC implements a word-level interleaving scheme to reduce banking conflict probability (typically 5% even for highly memory-intensive applications). The cores share a 2-level latch-based instruction cache: the first level (512 B) is private, the second level (L1.5) is a 4 kB 8-banks shared cache connected to the L1s with an interconnect similar to the LIC. The L1.5 refills from a larger 80 kB L2 memory hosting resident code. A dedicated hardware block (Event Unit) assists the cores to accelerate parallel computation patterns, such as thread dispatching and barriers. Finally, the SoC includes a controlling RISC-V core, a set of standard peripherals, and 3 FLLs for frequency control.

4.5.1.1 Dynamic Bit-Scalable Execution Processor

The proposed processor extends RI5CY, a 32-bit 4-pipeline stages core featuring DSP extensions such as 16-bit and 8-bit SIMD dot product fully supported by a GCC 7.1.1 toolchain [25]. The key efficiency-boosting enhancement is a new mixed-precision SIMD dot product execution unit, shown in Fig. 4.14. It includes 4 multiplexed sub-units implementing 16b down to 2b dot products (DOTP). To enable any SIMD mixed-precision computation, a slicer-and-router unit selects the correct bits in the source registers and forwards them to the DOTP unit featuring the higher precision between the two operands after optional bit manipulation. A dedicated circuit gates the clock of

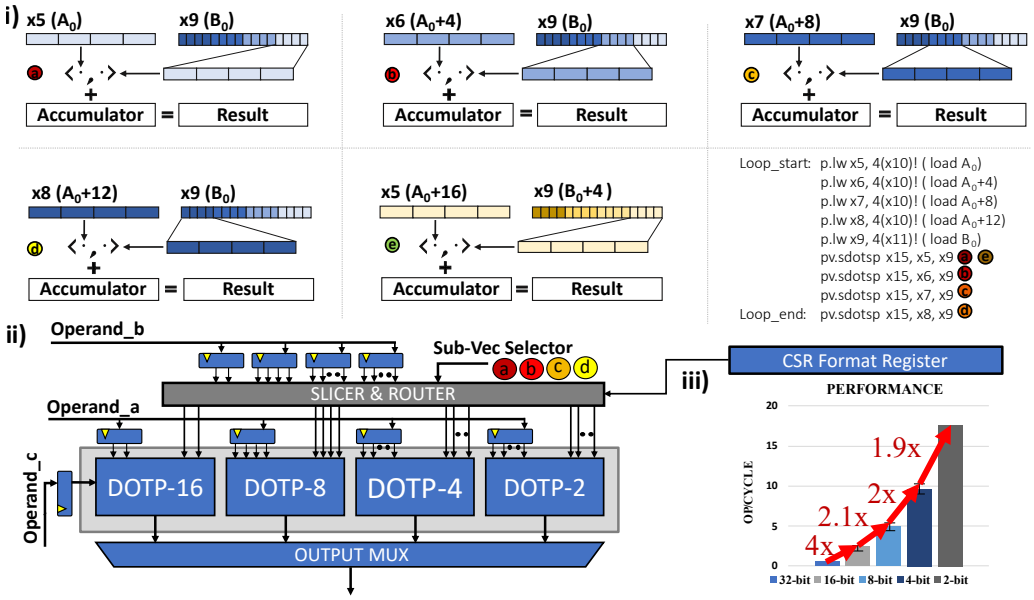


Figure 4.14: i) Mixed-Precision Dot Product 8x2; ii) Dot product functional Units; iii) Performance spanning through bit-widths.

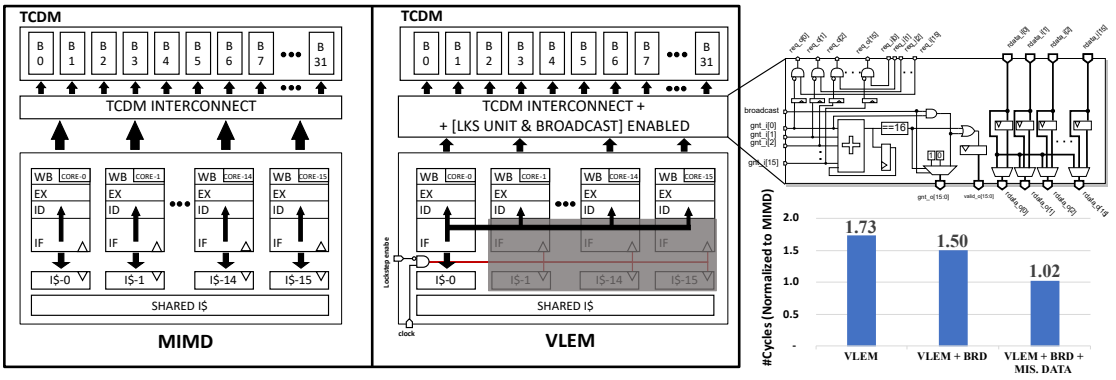
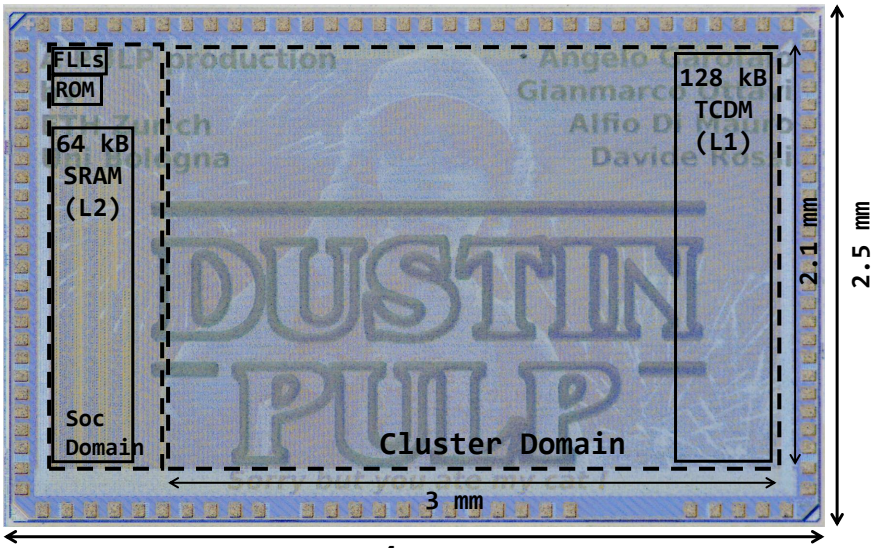


Figure 4.15: Overview of the cluster architecture to operate in VLE mode and comparison with the classic MIMD mode. The chart (bottom right) shows the optimizations to reduce the VLE execution overhead: First we introduce the broadcasting feature (+ BRD), then we operate the misalignment of the data (+ MIS. DATA).

the input registers of the unused SIMD units. With no timing overhead and an increase in area smaller than 10% with respect to RI5CY, the proposed power-aware design allows the extended core to run in the same power envelope as the original one, safeguarding its general-purpose computing efficiency. To encode the new mixed-precision SIMD instructions, we define a *virtual instruction*: the opcode (e.g., dotp) is decoded in the ID stage, the precision of its operands (e.g., 4x8) is specified by a control and status register (CSR), written by the processor before issuing a portion of code containing virtual SIMD instructions. This approach is essential to address the saturation problem of the RISC-V encoding space, as it avoids to explicitly encode all the 500 combinations of mixed-precision operands.



Technology	Chip Area	Total SRAM	VDD range	Freq. Range	Power Env.
CMOS 65nm	10mm²	208 kB	0.8 – 1.2V	60–205 MHz	156 mW
Int. Perf. (8b)	**Int. Eff. (8b)**	**Int. Perf. (4b)**	**Int. Eff. (4b)**	**Int. Perf. (2b)**	**Int. Eff. (2b)**
15 GOPS	303 GOPS/W	30 GOPS	562 GOPS/W	58 GOPS	1.15 TOPS/W

Figure 4.16: Chip micrograph and specifications.

4.5.1.2 Vector Lockstep Execution Mode

The second key efficiency enhancement is at the cluster level: we support a new Vector Lockstep Execution Mode (VLEM), where all cores execute the same instructions cycle-by-cycle. In VLEM, only the master core's L1 cache and IF stage are active, forwarding instructions to the ID stages of all cores (Fig. 4.15). The related activity reduction by clock gating saves up to 38% total power. To enter in VLEM, all cores have to i) synchronize on a barrier, ii) write to a memory-mapped register. Banking conflicts on TCDM are solved by delaying the grant signal assertion for the time required to serve all requests. To avoid systematic conflicts (e.g., when all cores access the same address in memory – a common pattern in linear algebra kernels), the VLEM unit is enhanced with a broadcast control, activated when all cores access the same memory location. Together with proper data organization, broadcast can entirely eliminate the overheads introduced by banking conflicts, as shown in Fig. 4.15, and can reduce the number of memory accesses up to 66%. After the execution of a kernel in lockstep, the cores exit VLEM by writing into a memory-mapped register. The increase in area of the slave cores (gating and isolation) is negligible (<3%) compared to the baseline as well as the design cost of the entire lockstep unit, which impacts for less than 1% on the total cluster area.

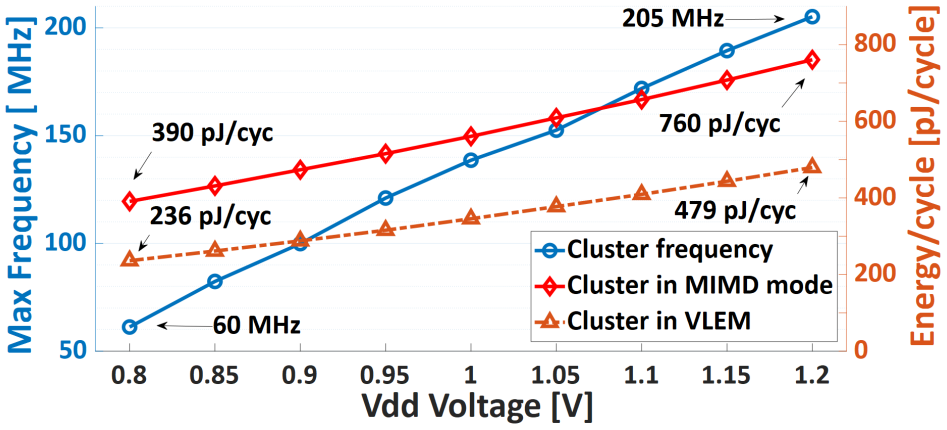


Figure 4.17: Voltage Sweep vs. Max Freq. vs. Energy/Cycle.

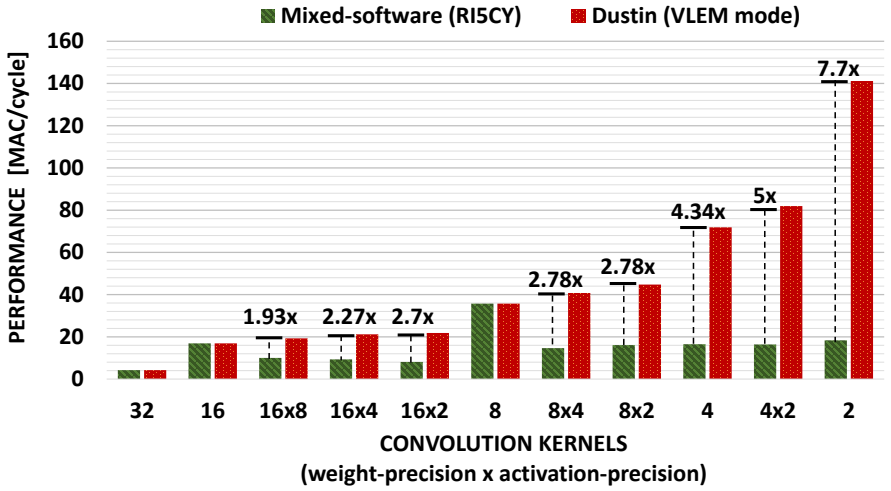


Figure 4.18: The chart compares the execution of mixed-precision convolution kernels running on the baseline 16 cores cluster with the RI5CY core (software mixed-precision kernels) and on Dustin's cluster in VLEM mode (featuring the Mixed-precision ISA extensions).

4.5.2 Measurements

Figure 4.16 shows a die photograph of DUSTIN, together with its main features. The SoC is implemented in 65 nm CMOS technology with a die size of 10 mm^2. Figure 4.17 reports the maximum operating frequency and the energy per cycle of the cluster over the 0.8V to 1.2V voltage range. The measurements are carried out on the silicon prototype, running a typical high-utilization deep neural network workload, the matrix-multiplication (matmul), with 8-bit precision operands. Linearly increasing with the voltage, we can reach the highest operating frequency of 205 MHz at 1.2V.

Figure 4.18 shows the performance of heavily quantized and mixed-precision convolutional kernels on the proposed cluster. On kernels where the activations are the only sub-byte precision operands, the performance benefits of the mixed-precision hardware

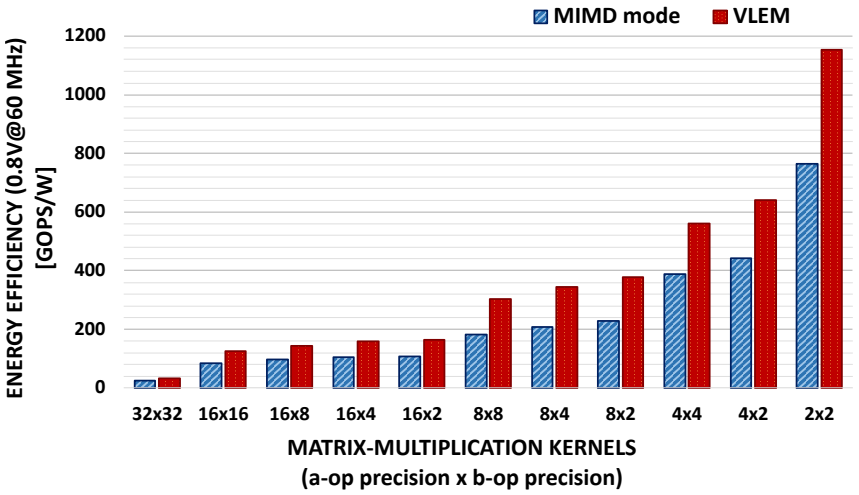


Figure 4.19: Comparison in terms of Energy Efficiency of Dustin configured in MIMD and VLE mode, running Mixed-precision Matrix Multiplication kernels.

Table 4.3: Comparison with SoA solutions.

	SleepRunner	**SamurAI**	**Mr. Wolf**	**VEGA**	**Dustin (this work)**
Technology	CMOS 28nm FDSOI	CMOS 28nm FDSOI	CMOS 40nm LP	CMOS 22nm FDSOI	CMOS 65nm
Die Area	0.68 mm^2	4.5 mm^2	10 mm^2	12 mm^2	10 mm^2
Applications	IoT GP	IoT GP + DNN	IoT GP + DNN	IoT GP + NSA+DNN	IoT GP + DNN + QNNs
CPU/ISA	CM0DS Thumb-2 subset	1x RI5CY RVC32IMFXpulp	9 x RI5CY RVC32IMFXpulp	10 x RI5CY RVC32IMFXpulp+SF	16 x MPIC CORES (RISC-V)
Int Precision (bits)	32	8, 16, 32	8, 16, 32	8, 16, 32	2, 4, 8, 16, 32 (plus Mixed-Precision)
Supply Voltage	0.4 - 0.8 V	0.45 - 0.9 V	0.8 - 1.1 V	0.5 – 0.8 V	0.8 - 1.2 V
Max Frequency	80 MHz	350 MHz	450 MHz	450 MHz	205 MHz
Power Envelope	320 µW	96 mW	153 mW	49.4 mW	156 mW
Best Integer Performance	31 MOPS (32b)	1.5 GOPS (8b) 2	12.1 GOPS (8b)	15.6 GOPS (8b)	15 GOPS (8b) 30 GOPS (4b) 58 GOPS (2b)
Best Integer Efficiency	97 MOPS/mW @ 18.6 MOPS (32b)	230 GOPS/W @110 MOPS (8b) 2	190 GOPS/W @ 3.8 GOPS (8b)	614 GOPS/W @ 7.6 GOPS	303 GOPS/W @4.4 GOPS (8b) 570 GOPS/W @8.8 GOPS (4b) 1152 GOPS/W @17.3 GOPS(2b)

extension are marginal due to the unpacking of data executed in a less arithmetic intensive portion of the kernel. In all other configurations, the mixed-precision instruction set extensions provide a significant advantage ranging from 2× to 7.7× improvements with respect to a baseline cluster.

To highlight the energy savings of the VLEM mode on regular computing kernels, we measure energy consumption with the cluster running the matrix-multiplication in two modes: the classic MIMD mode and the VLEM mode, enabled via software. Fig. 4.19 shows the related efficiency. The execution of linear kernels in VLEM mode achieves 1.5× better energy efficiency and no performance overhead with respect to the default MIMD execution.

Table 4.3 shows a comparison with the SoA. Compared to similar fully programmable IoT end-nodes [24, 87–89], the proposed SoC delivers similar performance and energy

efficiency on 8-bit format, despite the less scaled technology node used for implementation. This is achieved thanks to the larger parallelism of the cluster and the VLEM mode saving up to 38% of overall power consumption. The proposed silicon prototype is the only one featuring support for fully flexible bit-scalable precision from 2b to 32b, improving performance and efficiency by 3.7x and 1.9x over the state-of-the-art (SoA) for heavily quantized and mixed-precision workloads, delivering a peak performance of 58 GOPS and a peak efficiency of 1.15 TOPS/W.

Chapter 5

Heterogeneous In-Memory Computing RISC-V Cluster

5.1 Introduction

The contribution presented in Chapter 4 brings the utilization of the core hardware resources up to 94%, close to the structural limit of the target in-order single-instruction-issue pipeline. The performance bottleneck, in that case, moves towards the interfaces between the core and the main memory. However, such scenario is common of traditional Von Neumann computing architectures and the cost of the data communication between the separated processing and memory units limits the performance and the efficiency achievable on such systems; this phenomenon is referred to as *Von Neumann bottleneck*.

The emerging Analog in-memory computing (IMC) paradigm promises to overcome this limitation by processing the data within the memory boundaries and shows one to several orders of magnitude improvements in terms of energy efficiency compared to MCU and digital ASIC solutions on MVM operations, core of AI workload [60, 63, 90]. The promised computing efficiency is especially appealing for modern TinyML tasks running on battery powered IoT devices.

Nevertheless, IMC accelerators are outstanding platforms to deploy MVM based operations but they can not sustain the heterogeneity of the IoT workload [60]. Hence, to target practical IoT applications IMC arrays must be enclosed in programmable heterogeneous systems, introducing new system-level challenges. This chapter aims at analyzing the system level challenges of integrating the IMC paradigm into heterogeneous systems and at giving insights to maximally exploit the opportunities of the AIMC technology, targeting extreme edge of IoT class of devices.

5.1.1 Motivation

AIMC performs data processing in situ within memory arrays. Matrix-vector multiplication (MVM) operands can be mapped on the cross-bars of a Non-Volatile (NV) memory array and the *dot product* operation is performed entirely in the analog domain, making IMC devices promising candidates to accelerate DNN workloads and overcome the well-known memory bottleneck affecting traditional AI digital accelerators [60].

Several demonstrations of AIMC-based architectures have appeared in the field of DNN inference acceleration, showing outstanding peak energy efficiency in the order of hundreds of TOPS/W [60, 63]. Industry interest in this technology is growing [91, 92]. From a research perspective, several prototypes claimed tens to hundreds of TOPS/W by exploiting many different approaches, with a quite diverse set of choices in terms of numerical precision and underlying memory technologies (both charge-based and resistance-based memory technologies can serve as elements for such computational units) [60, 63].

However, several fundamental challenges are still open to achieve the claimed levels of efficiency at full-application scale: the intrinsic variability of analog computing both in the charge-based and resistive domain [63]; difficulties in dealing with low-precision computations that are often the only ones supported by AIMC-based architectures [63]; the necessity of specialized training [93]. Most prominently, a key issue is the limited flexibility of IMC arrays, which are extremely efficient on MVM or similar vector operations, but they are not flexible enough to sustain other types of workloads. To tackle this limitation, a prominent solution is to couple either general-purpose processors [94] or specialized digital accelerators [61] with analog in-memory computing cores. This allows extending the functionality of In-Memory Accelerators (IMA), creating heterogeneous analog/digital computing fabrics, connected to the system bus [94]. However, this integration poses severe concerns at the system level, mainly on two aspects: bandwidth and flexibility.

First, IMC acceleration moves the challenge towards ensuring efficient data movement within the system. In the case of volatile technologies, such as SRAM-based IMC, the weights of the DNN must be stored in non-volatile memory (external or internal to the system). This requires additional energy and time to move the data that must be stored into the cells of the IMA, anytime the cross-bar is programmed [60]. When considering non-volatile technologies, such as Flash, ReRAM or PCM-based IMC arrays, weights are directly stored into the cross-bar, with no need for marshaling operations. However, previous concerns continue to affect the activations that must be moved at the boundaries of the IMC array, to perform MVMs. Taking this into account, efficient

integration of IMC into heterogeneous systems requires an optimized interface design between the highly parallel IMC inputs/outputs, the programmable cores, and the rest of the system: low bandwidth and high communication latency between the processor and the IMA might create a major bottleneck [94].

Second, as a consequence of Amdahl's effect, accelerating MVM operators with an IMA moves the performance bottleneck on all the other computation needed to accomplish a certain task, which must be performed on the digital part of the system. Complex real-world neural networks mix MVMs with other workloads such as residuals, activation functions, or depth-wise convolutions; coupling the IMA with a single core, as has recently been proposed [94], will likely hit Amdahl's effect caused by the single-core bottleneck, hindering the whole computation performance. This chapter addresses the system-level challenges of analog IMC by exploiting extreme heterogeneity.

5.1.2 Contributions

The main contributions are the following:

- The design of a heterogeneous tightly-coupled shared-memory cluster that integrates 8 fully programmable RISC-V processors, an analog in-memory computing accelerator (IMA), and a dedicated digital block to accelerate depth-wise convolutions; A post place&route silicon-ready implementation targeting GlobalFoundries 22nm FD-SOI technology;
- The optimization of the interfaces between the analog IMA and the rest of the system to match the computing and IO requirements of the IMA, achieving performance as high as 958 GOPS on MVMs, more than 90% of its peak theoretical throughput, surpassing by one order of magnitude other approaches where the IMA is connected through a low-bandwidth, high-latency system bus [94];
- A deep analysis of the system benchmarking on a *Bottleneck* layer, representative of modern DNNs exploiting heavily heterogeneous layers such as point-wise, depth-wise convolutions and residuals, in terms of performance and energy efficiency. The analog/digital synergistic approach demonstrates full mitigation of Amdahl's effect, showing 2.6× better performance and 2.8× better energy efficiency compared to executing the layers on previous work that integrates only 8 programmable cores and the IMC analog array [95];
- The scalability of the previous architecture to a IMC multi-array system to analyze the challenges and the hardware resources necessary to enable end-to-end inference of a MobileNetV2. The architectural paradigm proposed executes inference

in $10ms$ with an energy of $482\mu J$, improving upon fully digital state-of-the-art solutions (SoA) [89] by $10\times$ in latency, reducing the energy consumption by $2.5\times$. Compared to SoA analog/digital architectures [94], the presented solution shows two orders of magnitude improvements in terms of execution latency.

5.2 Related Work

Charge-based memory technologies (e.g. SRAM [59], Dynamic Random Access Memory (DRAM), Flash) and non-volatile (NV) resistive memory technologies [96] (e.g. Resistive Random Access Memory (ReRAM) [97] Phase-Change Memory (PCM) [63] and MRAM [98]) both serve as computing substrates for analog in-memory computing. In this section, we review the State-of-the-Art (SoA) advancements in in-memory computing technology, circuits, and systems.

5.2.1 IMC Arithmetic

Low-bit-width integer computation is widely adopted in edge Artificial Intelligence (AI) applications, because of its higher efficiency and lower hardware cost than floating-point. In the IMC domain, the advantages are even more evident. Low bit-width data representation results in less area and power costs to design analog to digital (ADCs) and digital to analog (DACs) converters, which are predominant in IMC arrays [60, 63]. The adoption of heavily quantized integer arithmetic (8-bit or less), especially for DNNs, is fully justified by the fact that Quantized Neural Networks (QNNs) show a negligible drop-in Top-1 accuracy compared to the full floating-point precision model, on many AI-enhanced edge applications [15]. Also, noise-robust networks are an active research field for IMC deployment [99].

5.2.2 SRAM technology

The Static Random Access Memory (SRAM) technology is the most mature one, optimized for decades to be used as volatile memory storage for digital computing architectures. SRAMs are used to perform MVM operations both in the digital and analog domains. In the digital domain, the computation is performed coupling SRAM cells with additional near-memory logic, such as elementary gates, full adders, or adder trees, building up a digital accelerator [100]. In the analog domain, SRAMs can map MVMs by exploiting capacitive charge redistribution mechanisms along the bit-lines of the memory array [63]. Compared to the analog approach, SRAM-based digital IMC provides

higher robustness to noise and process, voltage, and temperature (PVT) variations, but significantly less advantages in terms of energy efficiency [101].

Most SoA academic SRAM-based IMC arrays operate in the analog domain [102]. One of the first prototypes appeared in 2018 [59], targeting binary-weight Neural Networks and demonstrating top-1 accuracy comparable with software accuracy on the MNIST dataset (∼98%). SRAM-based IMC has been demonstrated for binary/ternary DNNs achieving 403 TOPS/W and software accuracy on ternary networks trained on the CIFAR-10 dataset [103], as well as for reconfigurable bit-precision MVM operations showing80 TOPS/W [104]. Other SRAM-based IMC architectures have been proposed, achieving similar accuracy and efficiency [105]. The major challenge at the circuits level, which is actively being investigated in the literature, remains the computation noise that limits the signal-to-noise ration, mainly due to the sensitivity to PVT variations [63], and non-linearities [60].

5.2.3 Resistive Memory technology

A new generation of IMC accelerators targets emerging resistive memory technology, driven by the much higher density scaling factor that these technologies offer compared to the SRAM [106]. Moreover, resistive memories such as ReRAM, Magnetoresistive Random Access Memory (MRAM), and PCM, show other important advantages: non-volatility, low power envelope, and multi-level storage [97]. IMC based on NV memories suffers from similar precision issues as SRAM-based IMCs, compounded by additional challenges coming from memristive devices, such as write variability and conductance variations (temporal and temperature-induced) [107].

From a system-level perspective, resistive memories serve not only as IMC primitives, but also as non-volatile storage blocks for DNN weights. This avoids moving weights across the system memory hierarchy, which is instead necessary for SRAM-based IMC. Contrarily to SRAM, re-programming the memristive cross-bars with new data during the network model execution is not affordable, due to the high latency and power consumption associated with re-writes of non-volatile memory cells, as well as their limited endurance (for ReRAM and PCM). This forces rethinking architectures as memory-centric, with additional digital logic around to perform ancillary operations, as is the focus of this work.

Considering ReRAM-based IMC, Chen *et al.* [97] demonstrate significant computing parallelism, performing 8k MAC operations simultaneously. Other works show ReRAM-based IMC arrays as dense as 2Mb [108] or 4Mb [109], with peak energy efficiencies in the range of 120-200 TOPS/W within a power envelope of few milliwatts, suitable for tiny

edge AI devices. Moreover, the IMC array in [108], integrated within a PCB hosting also an FPGA, runs a ResNet-20 trained on the CIFAR-10 dataset with 90% top-1 accuracy.

For the MRAM technology, Doevenspeck *et al.* [98] present a Spin-orbit Torque MRAM (SOT-MRAM)-based IMC macro and demonstrate for the first time that resistive MRAM devices can be used for DNN applications. They claim software-like accuracy on a network targeted to the MNIST dataset.

PCM-based IMC arrays have been applied in mixed-precision in-memory iterative computing, combining a computational memory unit to perform the bulk of a computational task, with a von Neumann machine, which implements a backward method to iteratively improve the accuracy of the solution. This approach has been demonstrated to solve linear equations [110] and in DNN inference and even training tasks [107], showing limited error in the computation and much higher efficiency compared to traditional approaches [63].

Khaddam-Aljameh *et al.* [46] recently presented a state-of-the-art 256×256 PCM-based IMC core targeting DNN inference, fabricated in 14nm, showing energy efficiency of 10.5 TOPS/W and performance density of 1.59 TOPS/mm^2 on inference tasks of multi-layer perceptrons and ResNet-9 models trained on MNIST and CIFAR-10 datasets, with comparable accuracies as software baseline. In this work, we adopt the PCM-based IMC presented in [46].

5.2.4 Architectures and Systems

As discussed, there are several challenges related to technology that affect both charge-based and resistive IMC circuits currently under scrutiny from researchers. However, provided that these issues can be solved, another essential challenge is the integration of in-memory computational primitives into heterogeneous systems. In this work, we focus in particular on this aspect.

IMC cores primarily target matrix-vector multiplications (MVMs) or other similar vector operations, showing incredible throughput and efficiency. Although MVM operations are predominant in modern DNNs, they still represent only a subset of the DNN computation [60], which also includes residual connections, pooling layers, non-linear activation functions, softmax, etc.. Increasing the throughput of MVMs with IMC moves the performance bottleneck to all the other layers, which can not be easily mapped on IMC arrays. From a broader application perspective, an edge-computing system might incur workloads characteristically different than MVMs, such as data management and control tasks that are performed together with neural tasks [111]. It is necessary, for a

complete architecture, to address this computation in a programmable way. This reasoning strongly motivates the integration of the IMC with other specialized accelerators and software programmable cores in heterogeneous architectures [60].

To the best of our knowledge, not many works specifically focused on the integration of IMC arrays in heterogeneous analog/digital systems have been presented in literature so far. Dazzi *et al.* [112] propose more advanced IMC multi-core approaches with very carefully staged core-to-core dataflow, but the focus is mostly on convolutions and there are no provisions for heterogeneous computing nor for computations that do not map efficiently on the AIMC arrays. Houshmand *et al.* [113] explore co-optimization strategies of IMC array size, memory hierarchy and data-flows to avoid efficiency degradation when the IMC core is integrated into a processing infrastructure including also memory buffers and small control units, but they do not investigate complex scenarios like heterogeneous systems.

Zhou *et al.* [61] propose a PCM-based IMC array modeled in 14nm technology, complemented with additional digital logic that performs activation and pooling operations. A small SRAM memory acts then as a layer-to-layer intermediate buffer, followed by a hardware block that handles IM2COL transformations. The proposed solution shows a peak 112 TOPS/W on MVMs and has been demonstrated on the execution of a custom DNN model, with 95.6% of accuracy, at a performance of 7.7 inf/s with 8.22 $\mu J/inf$. However, this type of architecture is not flexible enough to support heterogeneous workloads, since it does not feature programmable cores.

Jia *et al.* [62] propose a 4×4 array of cores consisting of charge-based IMC crossbars extended with a programmable near-memory-computing (NMC) digital accelerator that performs single-instruction-multiple-data (SIMD) computing, shifting, pooling, and activation functions. On 8-bit MVMs, the prototype in 16nm shows 3 TOPS of peak performance with 30 TOPS/W of efficiency. Coupled with off-chip FPGA and MCU that handle communication with a host PC and control flows, the prototype has been demonstrated on a ResNet-50 model with 4-bit weights and activations, achieving a peak performance of 3.4 TOPS.

The silicon prototype presented in [94] integrates a charge-domain compute-in-memory unit supporting 1to8-bit×1to8-bit matrix-vector multiplications, into a tiny RISC-V CPU enriched with a direct memory access controller (DMA) and a set of peripherals. It shows a peak efficiency of 400 TOPS/W on the end-to-end inference of a binarized ten-layers network trained on CIFAR-10. However, also in this case the architecture can not afford complex heterogeneous computation: the core delivers only a

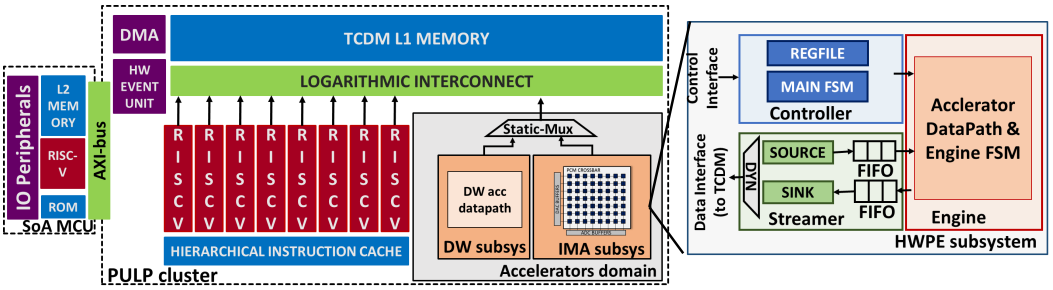


Figure 5.1: Overview of the PULP cluster architecture, integrating the IMA and the digital depth-wise accelerator. Each accelerator is enclosed into a Hardware Processing Engine (HWPE) subsystem, depicted on the right.

few million operations per second and it can only be used for control tasks such as programming DMA transfers, not being capable of performing compute-intensive functions with sufficient performance level.

The limits of the above-mentioned systems are mainly two: the integration of the IMA is loosely-coupled, with the IMA connected with other cores through a low bandwidth, high latency system bus; the presented heterogeneous systems are demonstrated either on neural networks model of few layers (trained on datasets such as CIFAR-10 or MNIST) or on custom NN models built ad-hoc to fit the requirements of the architecture. Neither approach is representative of modern DNN models widely used in classification and detection tasks at the edge of the IoT.

To overcome these limitations, this chapter presents an IMA coupled with a novel design of a digital accelerator to improve the efficiency of depth-wise kernels, integrated into the heterogeneous system, which is fully implement with the GlobalFoundries 22nm FD-SOI technology. Furthermore, the architecture is scaled up to explore the resources necessary to enable the end-to-end inference of a full MobileNetV2 network, a much more realistic benchmark for the class of networks that an ultra-low-power IoT end-node could target.

5.3 Heterogeneous Cluster

This section presents the analog IMA, the depth-wise digital accelerator, and their integration into the PULP cluster through a standardized interface called Hardware Processing Engine.

5.3.1 Hardware Processing Engines

To improve the performance and the energy efficiency of the accelerators in data movement operations, and to ease their integration into the cluster, each of the two accelerators presented here is incorporated as a Hardware Processing Engine (HWPE) using a standardized interface[1]. HWPEs expose a control and a data interface towards the rest of the cluster. The control interface allows the cluster's cores to access the registers of the targeted accelerator for configuration. The data interface is connected to the TCDM memory through multiple master ports on the logarithmic interconnect, similarly to what happens with the cores of the cluster. The width of this bus is a design-time parameter and can be chosen depending on the required bandwidth of the accelerator.

Fig.5.1 shows the heterogeneous cluster with two distinct HWPE interfaces encapsulating the IMA (namely *IMA subsystem*) and the depth-wise accelerator (namely *DW subsystem*). To avoid a large increase in the area of the logarithmic interconnect and in the latency of its arbitration scheme, the data interface of the *IMA subsystem* and the *DW subsystem* are statically multiplexed towards the TCDM, sharing the same physical ports on the interconnect. The two accelerators are used in a time-interleaved fashion, allowing one accelerator to full access the TCDM at a time. This choice does not cause any performance degradation, since in our DNN computing model the depth-wise accelerator and the IMA can not be active concurrently. However, they can be programmed independently and in parallel by the cores of the clusters. Each accelerator has its own programming bus and the configuration registers are mapped in different regions of the cluster memory map. To ease the programming phase of the accelerators, a set of Hardware-Abstraction-Layer (HAL) functions are exposed to the programmer and can be inferred directly into the C code through their explicit invocations. To reduce the power consumption of the cluster on jobs deployed to HWPEs, the latter expose an end-of-computation signal towards the cluster. After programming the HWPE and triggering its execution, the cluster cores can enter a low-power clock-gated sleep mode. Once the HWPE notifies an end of computation signal, the core can be woken up by the cluster Event Unit.

From the inside, HWPEs consist of three main blocks: the *Controller*, the *Engine*, and the *Streamer*. The *Controller* contains a memory-mapped latch-based register file used to store the configuration of the execution of the accelerator, and the main *Finite-State-Machine* (FSM) of the HWPE system, that coordinates the other blocks. The *Controller* can be targeted by the cores of the cluster in a memory-mapped fashion via the control interface introduced above. The semantic and the number of registers

[1] https://hwpe-doc.readthedocs.io/en/latest/

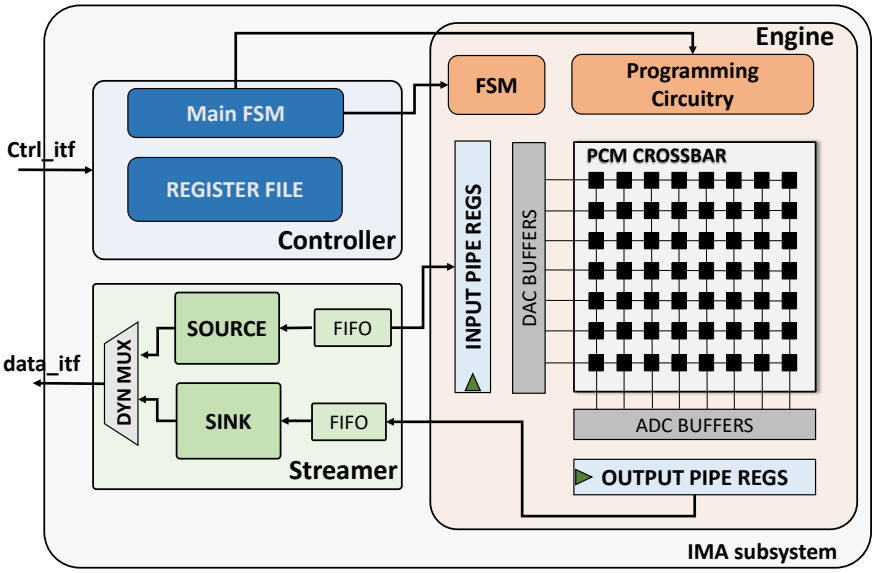


Figure 5.2: IMA enclosed into the HWPE interface. The *data_itf* width is designed to match the IO requirements of the IMA.

as well as the FSM are customized to accommodate the requirements of the enclosed accelerator. The *Engine* contains the data path of the accelerator and the specific FSM that coordinates the execution flow. It is, therefore, highly dependent on the specific accelerator design.

The *Streamer* contains the blocks necessary to move inputs and results in and out of the accelerator through its master port of the data interface and transform the memory accesses into coherent streams to feed the accelerator *Engine*. The streams are organized in two separated modules, namely *source* for incoming streams and *sink* for the outgoing ones. Both *source* and *sink* include address generators capable to generate three-dimensional access patterns in TCDM with configurable strides. They also include a re-aligner module to form word-aligned streams from non-word-aligned memory accesses, without constraining the memory system outside the HWPE to support misaligned accesses. The memory accesses generated by the two streams are dynamically multiplexed towards the data interface. Such a choice avoids the duplication of the data interface ports while not causing any performance overhead; eventual contentions are efficiently solved by an arbiter featuring a round-robin arbitration policy. Intermediate FIFOs in both directions are used to decouple the streams from memory contentions stalls and reduce the pressure on timing closure of the tightly-coupled system.

5.3.2 In-Memory Computing Accelerator Subsystem

Fig. 5.2 shows the integration of the IMC cross-bar within the HWPE. The width of the IMA data interface is sized to sustain the bandwidth requirements of the analog

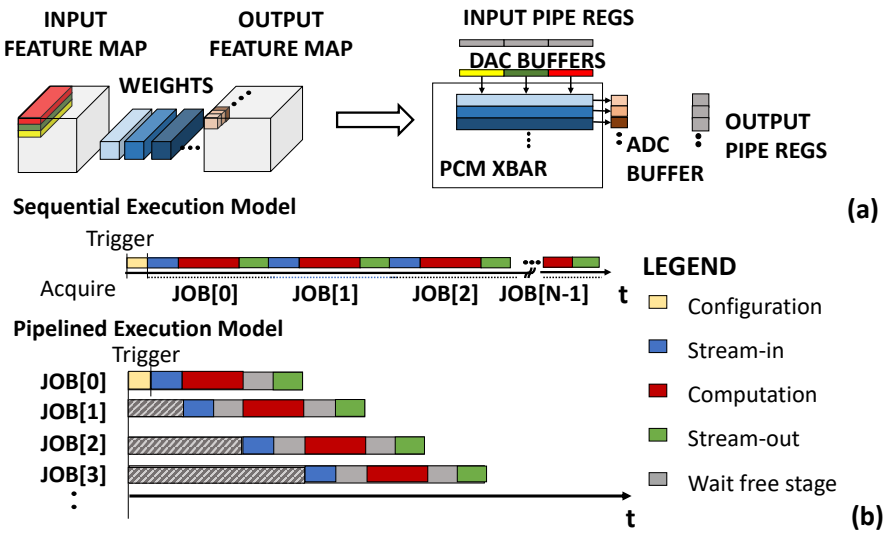


Figure 5.3: (a) Mapping of standard convolutions on the PCM crossbar. (b) Timeline of the sequential and pipelined execution models.

core, as shown in Sec. 5.4.2. The *Engine* contains both the digital and analog parts of the IMA data path. The digital part is composed of buffers for ADCs and DACs and of control circuitry; the analog core encloses the PCM devices (including PCM programming circuitry), and the ADCs and DACs themselves.

The IMA works on input data stored in L1 with the HWC format, i.e., with consecutive data elements encoding pixels that are adjacent in the channel dimension. Fig. 5.3 shows how a CNN layer is mapped on the IMA array. For a standard convolution, the streamers can directly perform a virtual IM2COL transformation [28], enabling to remap the computation to matrix-vector products of the form discussed in Sec. 2.4. As a consequence, the PCM array computes C_{out} channels of one output feature map pixel from a complete input volume of $C_{in} \times K \times K$ pixels in a single operation (that we call *job*), where C_{in}, C_{out} indicate the number of input and output channels, and K is the filter size.

The configuration sequence of the IMA starts when a core acquires a lock over the accelerator by reading a special ACQUIRE register through the control interface. After that, the core can interact with the IMA by programming the PCM devices with the weights of one or multiple layers; reading the conductance value of a PCM device; configuring a *job* setting the address of input and output data in TCDM and the ADC configuration. When the configuration ends, the execution can be started by writing to a special TRIGGER register. To minimize the IMA configuration and synchronization overhead, multiple jobs can be pipelined by setting the register file with the correct strides. In this way, a whole layer can be executed with only one configuration phase.

We propose two execution models for back-to-back job operations of the IMA: a simpler one, sequential, and a more optimized pipelined execution model. The relative timelines are shown in Fig. 5.3. The sequential model splits the execution of the single job into three phases operated sequentially. STREAM-IN: fetch data from the TCDM that is then streamed to the engine's internal DACs buffers; COMPUTATION: analog computation on the crossbar and writing of the ADCs buffers; STREAM-OUT: stream data from buffers back to the TCDM. In Sec. 5.4.2, we study how this model quickly becomes a bottleneck for the IMA's peak performance.

In the pipelined execution model, the three aforementioned phases of different jobs can overlap each other at the cost of additional hardware resources: we add two pipeline registers before and after the DACs and ADCs buffers and we extend the *Engine* FSM with additional states to control the overlapping phases: during the computing phase of the $i-th$ job (if not the last one to compute), the *engine* FSM sets the streamer to start a new memory transaction to fetch the inputs for the successive $(i+1)-th$ job. When such stream-in phase has finished, if there are the results of the previous job $(i-1)-th$ to stream-out, the *engine* FSM can configure the stream, as shown in Fig. 5.3. If we consider only the digital logic of the accelerator around the IMA, the pipelined approach increases the area by about 40%, due to the doubled number of input/output registers needed to enable the pipeline. However, this overhead reduces to 5% if we consider the total area of the accelerator (digital logic and analog IMC cross-bar), compared to the sequential approach.

5.3.3 Specialized Digital Accelerator

Depth-wise (DW) convolutions have been introduced in SoA DNNs such as MobileNetV2 to shrink the model size of the neural networks (by 7 to 10×) and their computational cost, with negligible accuracy drop [114]. Due to their lower connectivity compared to standard convolutions (each output channel depends only on a single corresponding input channel), DW layers are generally inefficient to map on IMC arrays, as we show in Sec. 5.4.2 on the *Bottleneck* use-case. Moreover, a pure software execution of such kernels easily becomes a performance bottleneck for computation [95]. To speed up the execution of the depth-wise layers, we therefore designed a specialized digital accelerator and integrated it into the heterogeneous cluster.

The accelerator we propose in this work is capable of processing depth-wise kernels on 8-bit signed input tensors and weights, accumulating the results in intermediate 32-bit registers and performing non-linear activation functions such as ReLU plus a set of ancillary functions (i.e. shifting and clipping) to bring back the final result into the 8-bit

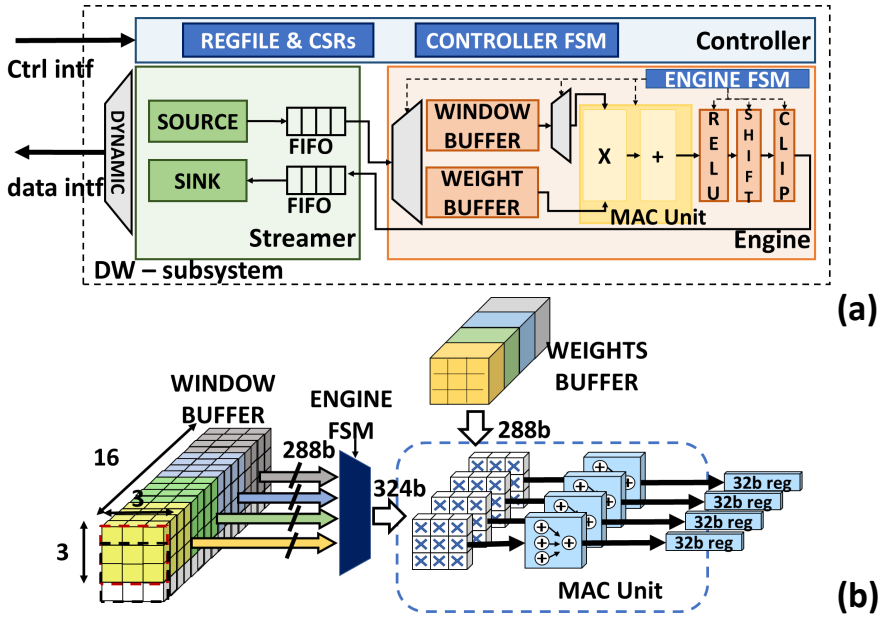


Figure 5.4: (a) Architecture overview of the Depth-wise digital accelerator, enclosed in the HWPE. (b) Execution flow of the depth-wise operation.

precision. Fig. 5.4 shows the general architecture of the proposed accelerator, enclosed in the dedicated HWPE interface.

The depth-wise accelerator employs a weight-stationary data flow and targets 3x3 depth-wise layers – the ones most commonly encountered in DNNs. The weights from 16 different filters, assumed to be 8-bit signed elements, are loaded from the TCDM memory at the beginning of the computation, sign-extended, and stored into a *weight buffer* that features 3x3x16 registers. The weights reside in the buffer until they have been used over the full input image. Input tensors are scanned by the accelerator using a vertically sliding window on the spatial dimensions, considering in each iteration 16 channels data stored in HWC layout (i.e., the same layout used by the IMA). The vertically sliding window is implemented utilizing a *window buffer* of 4x3x16 8-bit registers: 3 rows to host the current window, plus 1 row of inputs being loaded concurrently with the current window computation. Other than the two buffers, the data path of the accelerator consists of a network of Multiply-and-Accumulate (MAC), and ancillary blocks to compute ReLu, shifting and clipping operations, as shown in Fig. 5.4. The MAC unit consists of 36 multipliers and a reduction tree that operate on a $3\times3\times4$ block of the window buffer, passed through the ReLu and shifting&clipping blocks, and stored in an *output buffer*.

Fig. 5.5a shows the details of the depth-wise accelerator datapath operation in the form of Python-like pseudocode. For a given block of 16 channels, the operation starts by preloading weights. At the start of each output column, the window buffer is loaded

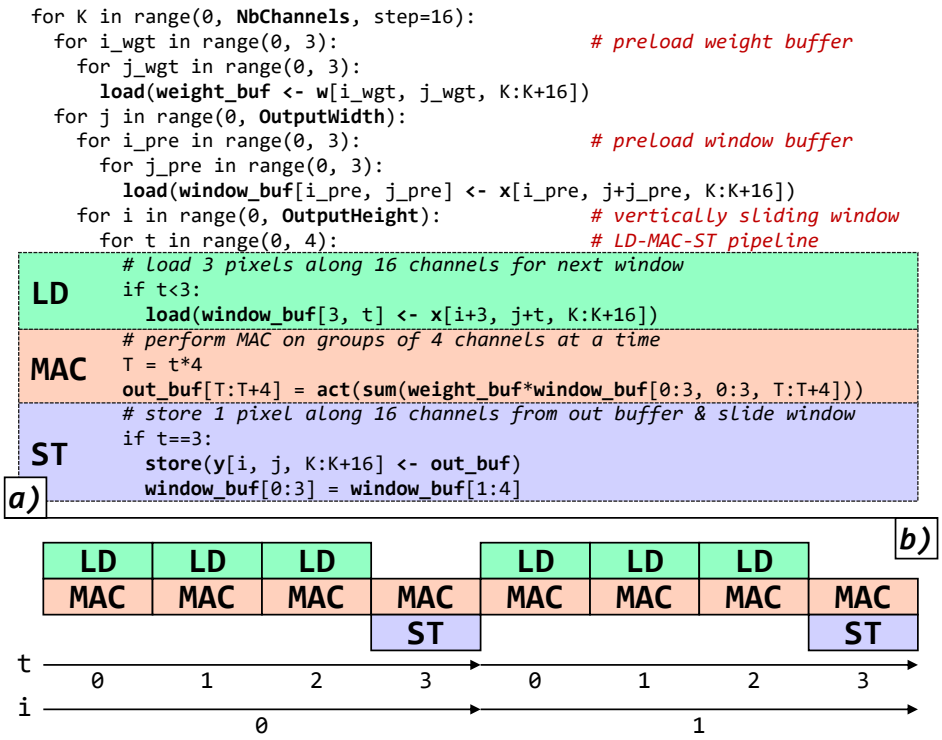


Figure 5.5: (a) Pseudo-Python code describing the operation of the depth-wise accelerator datapath. (b) Detail of the *LD* - *MAC* - *ST* pipeline.

with the content of the first 3x3 window; then, the operation of the datapath is organized in three pipelined stages, active over an inner loop of 4 cycles as shown in Fig. 5.5a and Fig. 5.5b. In the first three cycles of the inner loop, the *LD* stage is active: one input pixel across 16 channels is loaded to fill the fourth row of the window buffer. The *MAC* stage is active in all cycles of the inner loop, working on 4 channels at a time. Finally, the *ST* stage is active only in the fourth cycle: during this stage, the content of the output buffer produced in the previous three cycles and the current one is streamed out of the datapath, and the window buffer slides one pixel down. In this way, during the main body of the computation, the accelerator fully exploits the available memory bandwidth of 16 Bytes per cycle and the HWC layout of data, which is advantageous because it is the same layout used by the IMA. Overall, the execution of a depth-wise layer on the dedicated accelerator improves by 26× over a pure software implementation, achieving an average performance of 29.7 MAC/cycle.

5.4 Results and Discussion

This section evaluates the proposed heterogeneous cluster. From a physical viewpoint, we analyze area, power and timing costs of the system. From a performance and

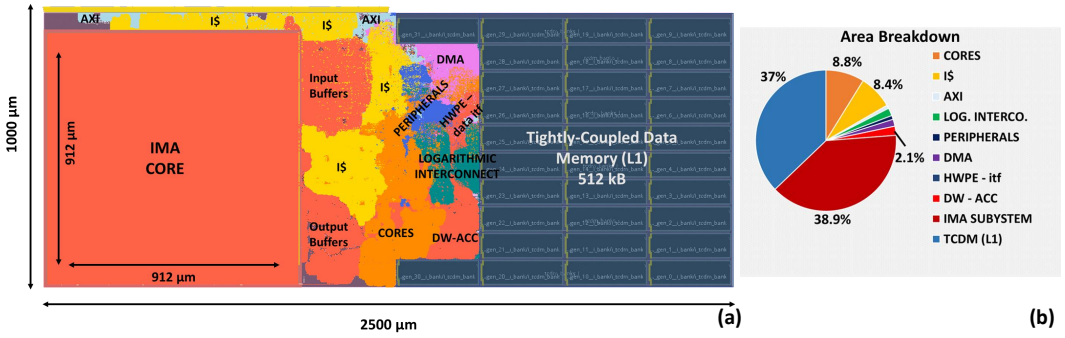


Figure 5.6: (a) Placed and Routed design of the heterogeneous cluster. (b) Area breakdown of the system.

energy efficiency viewpoint, we report the results of benchmarking hetereogeneous DNN layers, such as the *Bottleneck*.

5.4.1 Physical Implementation

To characterize the system in terms of area, power, and performance, we implement the cluster using the GlobalFoundries 22nm FD-SOI technology node. We synthesize the heterogeneous cluster with Synopsys Design Compiler-2019.12 and we perform a full place&route flow using Cadence Innovus 20.12, targeting the worst-case corner (SS, 0.72V, -40°/125°). The floorplan of the system is reported in Fig. 5.6. The analog IMC accelerator models, validated on silicon and modeled through silicon characterization of 14 nm prototypes, are fed into technology libraries (*.lef*, *.db*, and *.lib*) integrated into the front-end and back-end flows of the system. The area, the timing, and the power consumption of the IMC accelerator are extrapolated from the on-chip measurements reported in [46], properly scaled to the 22nm technology node. The power scaling is done according to the classical scaling theory under constant frequency, scaling power by $a \cdot b^2$, where a denotes the dimensional scaling and b is the voltage scaling factor. The area scaling follows the dimensional scaling. We assume that the IMA latency will remain constant. The total area of the heterogeneous cluster is 2.5 mm^2, partitioned among the several hardware blocks as shown in Fig. 5.6(b). As expected, the IMA subsystem and the 512 kB of TCDM memory occupy the major part of the total area (∼1/3 IMA, ∼1/3 TCDM, 1/3 the rest of the cluster), while the depth-wise accelerator has a negligible impact (2.1 %). The maximum operating frequency achievable by the final design is 500 MHz.

To perform power measurements we run parasitics-annotated gate-level netlist simulations of the digital part of the system in the typical corner (TT, 25C) at the operating voltage of 0.8V, executing DNN layers introduced above. The VCD simulation traces

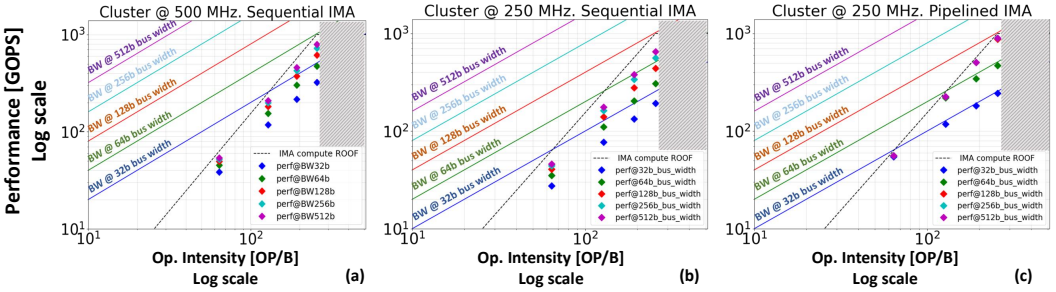


Figure 5.7: Roofline model of the IMA heterogeneous system. The compute roof of the IMA is a diagonal line, which depends quadratically on the operation intensity, not on the cluster frequency. The intersection of a bandwidth line with the compute roof defines a region where the performance points can lay for that configuration. In (a) and (b) cluster is running at 500MHz and 250MHz, respectively, with sequential execution of IMA. In (c) it runs at 250 MHz with a pipelined execution model for the IMA.

are analyzed with the Synopsys PrimeTime tool and the extracted power is integrated with the power extrapolated for the IMC accelerator [46].

Hence, the results presented in this section and in the following ones 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. We emphasize that neglecting these factors would cause significant underestimations in the clock tree dynamic power.

5.4.2 In-Memory Computing Accelerator Performance

First, we analyze the peak performance achievable by the IMA. An important point to stress is that, in contrast with digital accelerators, the maximum performance of the IMC array is only related to its MVM operation latency and its size (256×256 in the context of this work). The peak throughput is 1.008 TOPS, given by the maximum number of operations (256 × 256 × 2 OPs) that can be executed in its latency of 130ns [46]. In practice though, the real throughput achievable is typically scaled by the utilization factor of the array: only if we map a 256 output-channel / 256 input-channel point-wise layer we can achieve the maximum utilization rate and, thus, throughput. Another factor that limits the real performance of the IMA sub-system is the memory bandwidth that the heterogeneous cluster can sustain to feed the IMA with new input data and to store the IMA results into the TCDM memory. If the computation is too fast compared to the stream-in and stream-out time, we lose performance because we are limited by the bandwidth of the system.

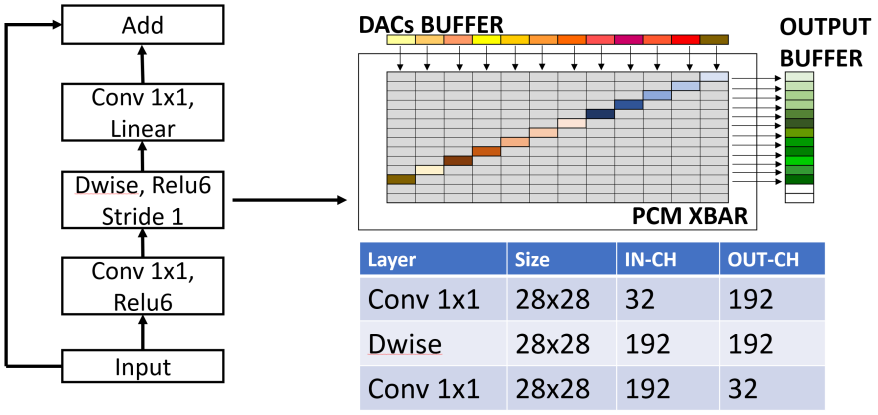
The PULP cluster can potentially offer high memory bandwidth towards the L1 memory thanks to the tightly-coupled interconnect scheme, at the cost of an increased

interconnect area (which scales linearly with the bit-width of the system bus), power, and timing. To find the width of the system bus able to sustain the IO requirements of the IMA at the lowest area overhead, we benchmark synthetic point-wise layers with different utilization rates of the IMC array (from 5% to 100%), varying the width of the IMA sub-system bus from 32-bit up to 512-bit.

In Fig. 5.7 we report the outcomes of our exploration as a roof-line plot [115]. The computing latency of the IMA does not depend on the cluster frequency, leading to two considerations: first, the compute roof of the IMA is a diagonal line proportional to the operation intensity (in other words, to the utilization factor of the IMA cross-bar) and not a line parallel to the x-axis, as is typically the case for digital systems; second, since the IMA computing latency is fixed, its memory bandwidth requirement change as we reduce or increase the cluster clock frequency. We investigate two operating frequencies, the maximum achievable one by the system when operating at high-voltage (500 MHz at 0.8V) and the maximum one achievable at low-voltage (250 MHz at 0.65V), and we compare the sequential and the pipelined execution models of the IMA.

In Fig. 5.7(a) the cluster is running at 500 MHz and we adopt the sequential execution model for the IMA. We observe that only with a 32-bit wide bus we are memory bound and a 64-bit wide data interface of the IMA subsystem is sufficient to fulfill the computing and IO requirements of the IMA. However, analyzing the performance at any of the system bus configurations above 32-bit we notice a gap between the compute roof and the real throughput, suggesting that we are under-utilizing the bandwidth of the system. The reason is that in the sequential model, as discussed in Section 5.3.2, 8% to 40% (depending on the size of the layer considered) of the total execution cycles are spent in stream-in and stream-out phases. In the rest of the execution, when the IMA is in the computing phase, the system bus is not solicited.

Analyzing the scenario where the cluster runs at 250 MHz we observe that higher bus-width (i.e. 128-bit) is necessary to preserve the peak performance of the IMA, as depicted in Fig.5.7(b). However, also in this case the sequential execution model is quite far from reaching the computing roof of the IMA. Despite the unavoidable area overhead compared to the sequential execution model (which, however, is limited to 5% if we consider the whole IMA sub-system, including the analog macro), Fig.5.7(c) shows that the pipelined solution guarantees full utilization of the bandwidth. At the system level, the optimal configuration is with a 128-bit wide system bus: using a narrower system bus, throughput is memory-bound, while using a wider one, performance does not improve, as computation is in a compute-bound region. In the optimal system configuration, the IMA can achieve a peak of 958 GOPS at 250 MHz, only 10% less



Layer	Size	IN-CH	OUT-CH
Conv 1x1	28x28	32	192
Dwise	28x28	192	192
Conv 1x1	28x28	192	32

Figure 5.8: Components of MobilenetV2 *Bottleneck* block with stride = 1 and mapping structure in the PCM crossbar for depthwise layers. All the gray rectangles are padding required for computing more than 1 channel per job.

than the theoretical peak performance at the compute roof, due to the programming overhead necessary to configure the accelerator and start the execution.

5.4.3 Case Study: The *Bottleneck* layer

To highlight the advantages, the trade-off, and the challenges of IMC on realistic use cases for edge computing and to assess the benefits of the presented heterogeneous system, we benchmark the *Bottleneck* layer of a MobileNetV2 DNN. We analyze three different computation mappings that are possible on the analog/digital system, compared to the baseline, which executes all the layers of the *Bottleneck* on the software cores using optimized software libraries [28]. The parameters of the selected *Bottleneck* layer are reported in Fig. 5.8: this configuration is chosen so that all the weights and activations fit the on-cluster TCDM (512 kB), without requiring any activation data tiling [38], the in-depth study of which is beyond the scope of the contributions of this chapter.

The first possible execution mapping is to offload all the layers of the *Bottleneck* to the IMA accelerator, except for the residual connection, which is always offloaded to the cluster's cores. To map the weights of the layers on the IMC cross-bar, we adopt the IM2COL approach [28]. Mapping point-wise layers is straightforward: each $1 \times 1 \times C_{in}$ filter is mapped across the height of the cross-bar (one column), more filters are mapped across the columns. If the layer parameters did not fit the size of the array, we would have to split the weights over multiple IMAs. We postpone the analysis of more complex scenarios, such as these, to Section 5.4.4 and we focus on the baseline case of a fully fitting layer here.

Contrarily to the point-wise layer, the depth-wise one is very inefficient to map on the IMA cross-bar. In depth-wise convolutions, each output channel depends only on the corresponding input channel: to make them suitable for mapping on the cross-bar array, a $K \times K$ kernel with C in/out channels must be expanded into a dense form, with all the weights out of a diagonal set to zero (padding), as shown in Fig. 5.8. Assuming a hypothetical IMC array large enough to fit all the weights and padding of the layer, out of $K^2 \times C^2$ crossbar locations programmed with weights or zeros, only $K^2 \times C$ of them would concur to the kernel computation.

To reduce the useless occupancy of crossbar cells (i.e. programmed with zeros), a different approach is to split the computation of C_{out} pixels, that normally would be computed in a single operation (what we call *job*), over multiple jobs, each of which computes $C_{job} < C_{out}$ pixels. As a trade-off, this leads to a smaller amount of operations per job, reducing the overall performance. The advantage of this method is that the total number of crossbar elements required to map the depth-wise kernel is $N_{xbar} = K^2 \times C \times C_{job}$, reducing the number of the overall programmed cells (with zeros and weights) by a factor of C_{out}/C_{job} compared to the previous approach, at the cost of additional $N_{jobs} = C_{out}/C_{job}$ jobs per output pixel to complete the execution of the kernel (note that in the previous case 1 job per C_{out} pixels is possible only on ideal infinite sized cross-bar).

Mapping all the layers of the targeted *Bottleneck* following the first approach is not feasible on the 256×256 cross-bar array we use: we would require 23× more cross-bar locations than the real number of weights, running out of IMC resources. Hence, we analyze the costs of separating the depth-wise in multiple jobs, considering two parameters: $C_{job} = 8$ and $C_{job} = 16$, which translates to an increase of 25% and 54% in the number of devices, respectively. Empirically, we consider these configurations as a reasonable trade-off between performance and occupancy of the cross-bar. The two configurations are referred to as IMA c_{job8} and IMA c_{job16}, respectively.

The second mapping we analyze executes the point-wise layers on the IMA and the depth-wise kernels on the 8 RISC-V cores of the cluster. The software kernels for the depth-wise are derived from an optimized parallel software library tailored on PULP-based clusters [28]. Since such kernels require the input data to be in Channel-Height-Width (CHW) layout and since the output from the point-wise layer (from the IMA) is in Height-Width-Channel (HWC) layout, additional execution cycles are needed for on-the-fly data marshaling operations. The output is generated in the HWC format instead and can be forwarded to the IMA with no additional overhead. This configuration is referred to as HYBRID and requires the storage of depth-wise weights in the TCDM, instead of in the IMA crossbar. This is a reasonable trade-off since the depth-wise

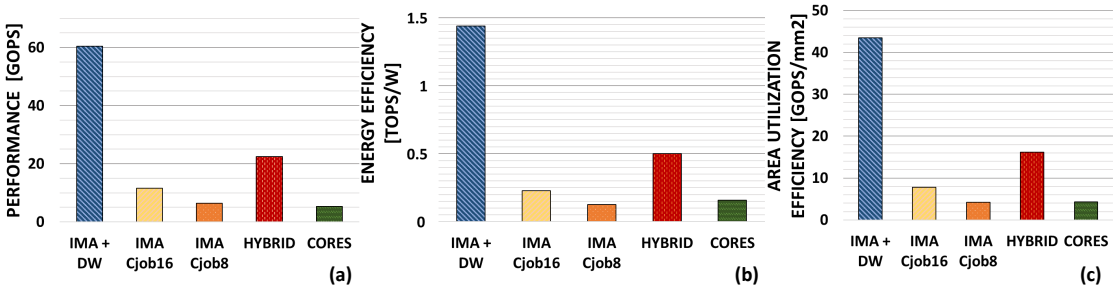


Figure 5.9: (a) Performance (in GOPS), (b) Energy Efficiency (in TOPS/W), and (c) Area Utilization Efficiency (in GOPS/mm^2) of the *Bottleneck* layer running on the cluster at 500 MHz with 128-bit wide system-bus. The area efficiency is related to the effective area of the PCM arrays utilized to implement the *Bottleneck* (including padding necessary to map the depth-wise on the IMA).

weights usually account for no more than 10% of the total of a depth-wise based neural network [114] ($\sim$ 4% in the considered *Bottleneck* layer).

The third mapping solution, indicated as IMA+DW, runs the point-wise layers on the IMA, the depth-wise layers on the dedicated digital accelerator, and the residual layer on the cores. The digital accelerator accepts input data and weights in HWC format and produces outputs in HWC format, requiring no additional data marshaling operations during the *Bottleneck* layer execution.

Benchmarking results are provided in Fig. 5.9 for all the solutions discussed above, in terms of performance, energy efficiency, and area utilization efficiency. The width of the system bus is 128-bit and we adopt the pipelined execution model for the IMA; as demonstrated in Sec. 5.4.2, this configuration maximizes performance. The cluster operates at 500 MHz at 0.8V, in typical operating conditions (TT, 0.8V, 25C).

We can notice that despite a significant area utilization of the IMC array, the performance of IMA c_{job16} and IMA c_{job8} are only 2.27× and 1.23× higher than a pure software execution of the *Bottleneck*. Efficiency is even worse: 1.23× lower energy efficiency and the same area efficiency of IMA c_{job8}, and comparable energy and area efficiency of IMA c_{job16} compared to the CORES demonstrate that IMC arrays are not efficient to host sparse layers like the depth-wise. The HYBRID solution instead achieves 4.6× better performance and 3.4× better energy efficiency than the CORES configuration. Despite software-based execution of depth-wise layers, this solution overcomes the c_{job16} configuration by 2× in terms of performance, by 2.3× in terms of energy efficiency, and by 2.1× in terms of area efficiency. The peak performance is achieved in the IMA+DW configuration, improving by 2.6× and 11.5× over the HYBRID and CORES solutions, respectively. By offloading point-wise layers to the IMA and depth-wise layers to the dedicated digital accelerator, we fully exploit the potential of the two hardware blocks, while the cores

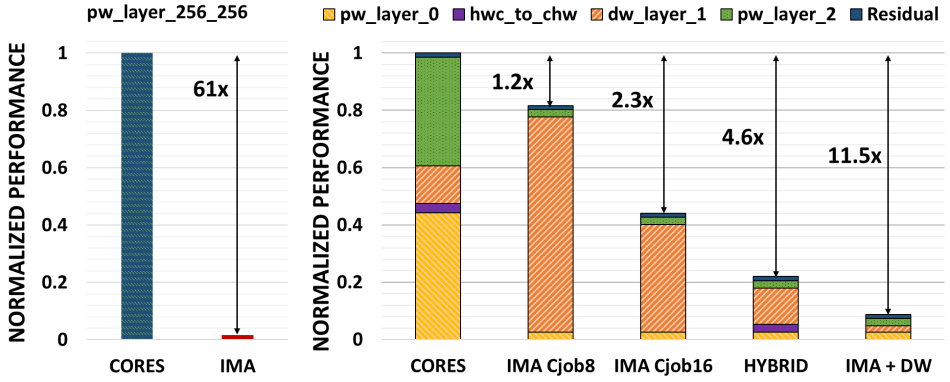


Figure 5.10: Normalized Performance, compared to full software implementation (CORES), of point-wise (left) and *Bottleneck* (right) layers. For the right-side analysis, the impact of each layer on the execution of the whole Bottleneck is shown, considering the different computing mapping solutions enabled by the heterogeneous cluster.

handle their configuration, the workload dispatching, and ancillary aggregation operations, such as the residual connection. This synergistic approach, enabled by the fact that cores, IMA, and depth-wise accelerator all share the same memory at L1, stands out also as the most efficient one, overcoming 2.7× and 9.2× the HYBRID and CORES configurations, in terms of energy efficiency, and by 2.5× and 10.2× the same configurations in terms of area efficiency.

Fig. 5.10 shows the execution breakdown of the *Bottleneck* layer. In a pure software execution scenario, the point-wise layers dominate the computation (CORES). The IMA shows significant acceleration in such dense MVM-based operations (left-sided Fig. 5.10), moving the performance bottleneck on other layers like the depth-wise. However, the IMA itself is not capable of mitigating this Amdahl's effect, since the execution of the depth-wise on the IMA is not efficient and dominates the total execution cycles (IMA c_{job8} and IMA c_{job16}). Execution of depth-wise convolutions on the cores (HYBRID) improves execution time, but this block remains by far the slowest one. On the other hand, offloading the depth-wise layer to the digital accelerator (IMA+DW) eliminates the performance bottleneck as the execution time of the depth-wise layer is comparable to the other components of the *Bottleneck*, such as point-wise layers and residuals.

5.4.4 End-to-End Inference of the MobileNetV2

In this section, we study the scalability of the heterogeneous system (in terms of challenges and hardware resources) to enable end-to-end inference of the MobileNetV2 [114]. To build the model of the scaled-up architecture, we start from the physical measurements carried out in the previous sections, introducing the following considerations: the PCM-based IMC cross-bar we use in this chapter does not support cell re-programming,

Algorithm 1 Full-Network Tile&Pack algorithm

function TILE&PACK($\mathbf{n}, \mathbf{h}, \mathbf{w}, S, n_{\text{ima}}$) ▷ $\mathbf{n}, \mathbf{h}, \mathbf{w}$ are name, height, width of all layers, S is the size of each IMA (default 256), n_{ima} is the number of available IMAs
  Tiles ← [ ]
  for all $n, (h, w) \in \mathbf{n}, (\mathbf{h}, \mathbf{w})$ **do** ▷ Create tiles
    $n_{\text{tile,w}} \leftarrow \lfloor w/S \rfloor$; $w_{\text{rem}} \leftarrow w \bmod S$
    $n_{\text{tile,h}} \leftarrow \lfloor h/S \rfloor$; $h_{\text{rem}} \leftarrow h \bmod S$
    for $i \in [0, n_{\text{tile,h}} - 1]$ **do**
      for $j \in [0, n_{\text{tile,w}} - 1]$ **do**
        Tiles$[n +$ "_tile i_j"$] \leftarrow (S, S)$
      end for
    end for
    for $j \in [0, n_{\text{tile,w}} - 1]$ **do**
      Tiles$[n +$ "_tile $n_{\text{tile,h}}$_j"$] \leftarrow (h_{\text{rem}}, S)$
    end for
    for $i \in [0, n_{\text{tile,h}} - 1]$ **do**
      Tiles$[n +$ "_tile i_$n_{\text{tile,w}}$"$] \leftarrow (S, w_{\text{rem}})$
    end for
    Tiles$[n +$ "_tile $n_{\text{tile,h}}$_$n_{\text{tile,w}}$"$] \leftarrow (h_{\text{rem}}, w_{\text{rem}})$
  end for
  for all $n, (h, w) \in$ Tiles **do** ▷ Remove 0-sized tiles
    if $h = 0$ or $w = 0$ **then** remove(**Tiles**$[n]$)
    end if
  end for;
  Bins ← BINBESTFIT(**Tiles**)
  IMA_Mapping ← MAXRECTSBSSF(**Bins**)
  return IMA_Mapping
end function

during the execution of the Neural Network model, due to the high latency of the operation. An iterative flow is necessary to program each cell of the PCM cross-bar: first, pulses are sent to the cell, then the conductance is read-out and compared with the expected value. The outcome discrepancy is used to modulate the successive pulses to repeat the procedure until convergence. The programming of the IMA is done in a diagonal [46] or row-wise [116] fashion, therefore takes considerably larger time (20× to 30× higher) than merely performing a parallel MVM. As a direct consequence, to map layers bigger than the cross-bar size we need to split the weights and the layer's execution on multiple IMAs, at the cost of additional area. However, having multiple IMAs allows reducing the occupancy of the generic memory of the system to store the weights, being them hosted by the cross-bar itself.

For this analysis, we integrate the IMC cross-bars into a single heterogeneous cluster of the same type presented in the previous sections. Specifically, multiple cross-bars are integrated into the same IMA sub-system, fully sharing the same data and control interface. They can be activated one at a time through a static multiplexing scheme. One multiplexer collects the data interfaces of the IMAs and redirects them into a 128-bit wide bus connected to the logarithmic interconnect of the cluster. However, they can all be programmed before the start of the computation, since we assume to replicate

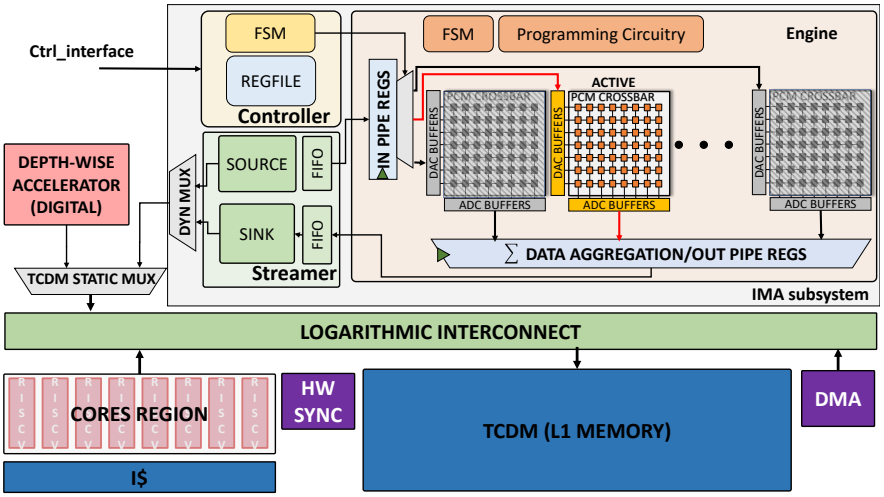


Figure 5.11: Overview of the scaled-up heterogeneous architecture. Only one IMC cross-bar can be active at a time.

the configuration registers (mapped in different portions of the cluster memory map). Fig. 5.11 shows an overview of the architecture.

We adopt a sequential execution model for the layer-to-layer inference of the network, with the additional condition that all the input activations reside in the L1 memory of the cluster. In our analysis, we do not directly consider the overhead in terms of time and energy to access activation data from on-chip memory hierarchies. Double buffering and activation data tiling have been shown to be effective at hiding the time overhead [38] and minimizing the energy one [89] in such cases, and we expect this effect to hold also in the case we analyze here. In the case of the considered MobileNetV2 we map only the point-wise layers on the IMA cross-bars, while the depth-wise ones are executed on the digital accelerator. As reported in Sec. 5.4.3, this solution leads to the highest performance and efficiency of the system.

Only the first layers of the MobileNetV2 fit a single 256×256 cross-bar, while the others (starting from the *Bottleneck 3*) require to be split over multiple IMA tiles. Therefore, we develop a TILE&PACK strategy, outlined in Alg. 1, to tile all layers and pack their contributions in the smallest number of IMAs. Tiling splits a layer over multiple IMAs only when it does not fit the size of the cross-bar; we do not allow tiling to fill unfilled IMA locations, aiming at the highest utilization area of the cross-bar on a per-tile basis. Packing is based on the *Maximal Rectangles Best Short Side Fit* bin fitting algorithm[2].

Fig. 5.12 shows the result of the application of the TILE&PACK algorithm to the weights of MobileNetV2. From this analysis, we conclude that to map all the *Bottleneck*

[2]We employ the open-source *rectpack* Python library, available at `https://github.com/secnot/rectpack` [117], to implement the BINBESTFIT and MAXRECTSBSSF functions.

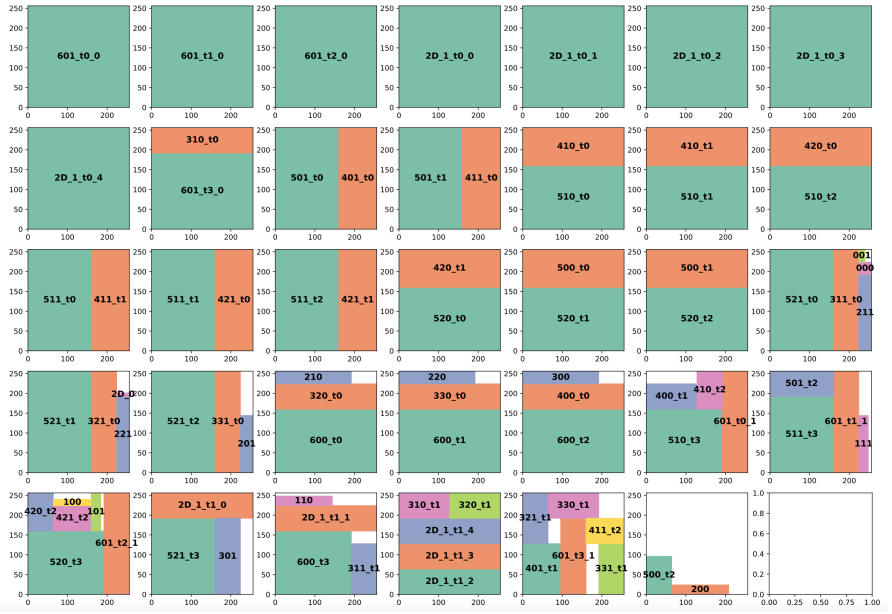


Figure 5.12: Mapping of the end-to-end MobileNetV2 on the 34 required IMAs, using the TILE&PACK strategy outlined in Alg. 1. The algorithm minimizes the number of IMAs necessary to map the NN model.

layers of the MobileNetV2 we need 34 IMA cross-bars. As can be seen in Fig. 5.12, the TILE&PACK algorithm achieves 100% of utilization of the cross-bar cells on most IMA cross-bars, with only the final one showing a utilization below 84%.

The system with 34 IMAs would require a minimum area of ∼30 mm^2, since the area of the single IMA is 0.83 mm^2. Despite this might represent a drawback, it is worth noticing that weights need anyway to be stored into a non-volatile memory inside or outside the system, such as a Flash. In principle, the non-volatility of PCM-based IMAs allows eliminating this Flash memory from the system.

Each layer or layer tile considered in this study is benchmarked in terms of execution latency and energy individually, on the heterogeneous system analyzed in Sec. 5.4.3 (which incorporates only one IMA). We argue that, for this study which aims to be a guideline for further digital/analog systems explorations, this is a good approximation, since the benchmarked results include input/output fetch/storage from/to the L1 memory of the system and the instructions of the cores to configure and start the execution of the accelerators (this reasoning holds for point-wise, depth-wise and residual connection layers).

The results are shown in Fig. 5.13, whereas in Fig. 5.14 we report the energy and the latency breakdown (among the several hardware blocks involved in the computation) of the conv-2d and *Bottleneck* layers. We notice higher execution latency and lower efficiency for point-wise layers with fewer parameters that operate on larger inputs –

LAYER		INPUT (H/W x C)	#PARAMS	MMAC	Latency [ms]	Energy [mJ]	GMAC/s/W
CONV - 2D	2D_0	224 x 3	864	10.84	1.63	0.042	255.6
0 - BOTTLENECK	000 - PW	112 x 32	1024	12.85	1.63	0.051	251.6
	DW	112 x 32	288	3.61	0.24	0.006	626.3
	001 - DW	112 x 32	512	6.42	1.63	0.044	147.4
1 - BOTTLENECK	100 - PW	112 x 16	1536	19.27	1.63	0.076	254.7
	DW	112x96	864	2.71	0.18	0.004	626.3
	101 - PW	56x96	2304	7.23	0.41	0.017	437.1
	110 - PW	56x24	3456	10.84	0.41	0.025	431.3
	DW	56x144	1296	4.06	0.27	0.006	626.3
	111 - PW	56x144	3456	10.84	0.41	0.020	540.4
	ADD	56x24	#:	0.08	0.18	0.006	12.1
2 - BOTTLENECK	200 - PW	56 x 24	3456	10.84	0.41	0.025	431.3
	DW	56x144	1296	1.02	0.07	0.002	626.3
	201 - PW	28x144	4608	3.61	0.10	0.005	684.7
	210 - PW	28x32	6144	4.82	0.10	0.008	612.8
	DW	28x192	1728	1.35	0.09	0.002	626.3
	211 - PW	28x192	6144	4.82	0.10	0.006	780.6
	ADD	28x32	#:	0.03	0.06	0.002	12.1
	220 - PW	28x32	6144	4.82	0.10	0.008	612.8
	DW	28x192	1728	1.35	0.09	0.002	626.3
	221 - PW	28x192	6144	4.82	0.10	0.006	780.6
	ADD	28x32	#:	0.03	0.06	0.002	12.1
3 - BOTTLENECK	300 - PW	28x32	6144	4.82	0.10	0.008	612.8
	DW	28x192	1728	0.34	0.02	0.001	626.3
	301 - PW	14x192	12288	2.41	0.03	0.002	1325.5
	310 - PW	14x64	24576	4.82	0.05	0.004	1117.6
	DW	14x384	3456	0.68	0.05	0.001	626.3
	311 - PW	14x384	24576	4.82	0.05	0.004	1325.5
	ADD	14x64	#:	0.01	0.03	0.001	12.1
	320 - PW	14x64	24576	4.82	0.05	0.004	1117.6
	DW	14x384	3456	0.68	0.05	0.001	626.3
	321 - PW	14x384	24576	4.82	0.05	0.004	1325.5
	ADD	14x64	#:	0.01	0.03	0.001	12.1
	330 - PW	14x64	24576	4.82	0.05	0.004	1117.6
	DW	14x384	3456	0.68	0.05	0.001	626.3
	331 - PW	14x384	24576	4.82	0.05	0.004	1325.5
	ADD	14x64	#:	0.01	0.03	0.001	12.1
4 - BOTTLENECK	400 - PW	14x64	24576	4.82	0.05	0.004	1117.6
	DW	14x384	3456	0.68	0.05	0.001	626.3
	401 - PW	14x384	36864	7.23	0.05	0.004	1727.3
	410 - PW	14x96	55296	10.84	0.08	0.007	1540.6
	DW	14x576	5184	1.02	0.07	0.002	626.3
	411 - PW	14x576	55296	10.84	0.08	0.006	1727.3
	ADD	14x96	#:	0.02	0.05	0.002	12.1
	420 - PW	14x96	55296	10.84	0.08	0.007	1540.6
	DW	14x576	5184	1.02	0.07	0.002	626.3
	421 - PW	14x576	55296	10.84	0.08	0.006	1727.3
	ADD	14x96	#:	0.02	0.05	0.002	12.1
5 - BOTTLENECK	500 - PW	14x96	55296	10.84	0.08	0.007	1540.6
	DW	14x576	5184	0.25	0.02	0.000	626.3
	501 - PW	7x576	92160	4.52	0.02	0.002	2280.4
	510 - PW	7x160	153600	7.53	0.03	0.003	2409.0
	DW	7x960	8640	0.42	0.03	0.001	626.3
	511 - PW	7x960	153600	7.53	0.03	0.003	2583.6
	ADD	7x160	#:	0.01	0.02	0.001	12.1
	520 - PW	7x160	153600	7.53	0.03	0.003	2409.0
	DW	7x960	8640	0.42	0.03	0.001	626.3
	521 - PW	7x960	153600	7.53	0.03	0.003	2583.6
	ADD	7x160	#:	0.01	0.02	0.001	12.1
6 - BOTTLENECK	600 - PW	7x160	153600	7.53	0.03	0.003	2409.0
	DW	7x960	8640	0.42	0.03	0.001	626.3
	601 - PW	7x960	307200	15.05	0.05	0.006	2583.6
CONV - 2D	2D_1	7x320	409600	20.07	0.06	0.008	2464.6
TOTAL			2190784	281.65	10.06	0.48	583.9

Figure 5.13: End-to-end execution of the MobileNetV2 on the scaled-up heterogeneous cluster. The figure shows the parameters of each layer, the execution latency (ms), the execution energy (mJ) and the energy efficiency (GMAC/s/W).

typically, the ones from layers appearing early on in the network. In these cases, the major part of the energy is spent in digital logic, as these layers require more input and output streams to move the activations to be processed. The most efficient layers are the last two, where the IMA is utilized best, showing a peak of efficiency higher than 5 TOPS/W. Overall, the proposed architecture executes the end-to-end inference in 10.1ms of latency, consuming 482 μJ.

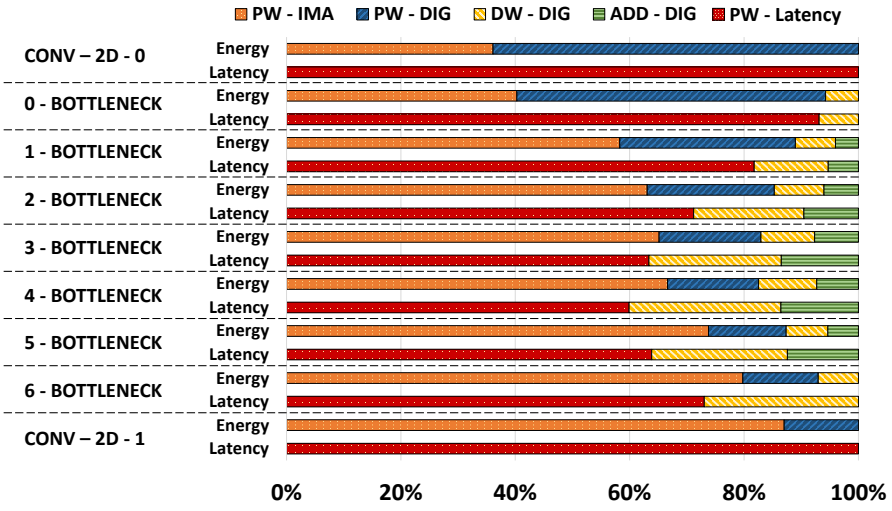


Figure 5.14: Latency and energy breakdown of *Bottleneck* layers of the MobileNetV2 executed on the scaled-up heterogeneous system.

5.4.5 Comparison with the *State-of-the-Art*

Tab. 5.1 reports the comparison of our scaled-up system with fully digital and mixed-signal state-of-the-art solutions. The solution we propose is superior compared to the others, as it provides full hardware support for a wide range of workloads both in analog and digital domains, enabling *de facto* efficient end-to-end execution of complex neural network models such as the MobileNetV2. Compared to Vega [89], which is an architecture based on the same RISC-V cluster without integrating analog IMC cores nor dedicated accelerators for the depth-wise, we show 10× and 2.5× improvements in terms of inference latency and energy, respectively, when considering the end-to-end inference of the MobileNetV2.

We compare favorably also with [94], which consists of a tiny RISC-V core and a charge-based IMC array integrated into the system through a loosely-coupled scheme. In theory, the presence of a programmable core potentially enables the execution of a reasonably sized network such as MobileNetV2. However, the only processing model possible on this architecture is to offload the point-wise layers to the IMC array and the depth-wise and residual layers to the tiny RISC-V processor, which is not capable of performing compute-intensive functions with a reasonable performance level. This would create a major performance bottleneck for the heterogeneous workload. For these reasons, our solution shows at least two orders of magnitude improvement on the end-to-end execution of the DNN. Despite the higher area of our system that might represent a drawback, it is worth noticing that the charge-based IMA integrated into [94] requires extending the architecture with a Flash memory to store the weights of the DNN (with

Table 5.1: Comparison with the State-of-the-Art.

	Rossi et al. [89]	**Zhou et al. [61]**	**Jia et al. [94]**	**Jia et al. [62]**	**This Work**
Tech. node	22nm	14nm	65nm	16nm	22nm
Area [mm2]	12	3.2	13.5	25	∼30
Cores (ISA)	9xRV32 IMCFXpulp	None	1xRV32 IM	None	8xRV32 IMCXpulp
Analog IMC	None	1024x512 PCM-based	2304x256 charge-based	1152x256 charge-based	34 256x256 PCM-based
Digital acc.	HWCE (standard conv.)	ReLu, activation processing, im2col	Activations, scalings, pooling	Activations, scalings, pooling	Depth-wise
Peak Perf. [TOPS]	0.032 (ML 8b)	2, 26.1 (8/4b-4b)	2.19 (1b-1b)	3 (8b-8b)	0.958 (8b-4b)
Peak Efficiency [TOPS/W]	0.61 (8b-8b)	13.5, 112 (8/4b-4b)	400 (1b-1b)	30 (8b-8b)	6.39 (8b-4b)
MobileNetV2 inference					
Inference Latency	10 inf/s	n/a	0.23 inf/s	n/a	99 inf/s
Inference Energy	1.19 mJ	n/a	n/a	n/a	0.482 mJ

[1] Scaled from 1b-1b MVMs performance as explained in [94].
[2] Point-wise latency estimated from the peak performance on 8-bit×4-bit MVMs. Latency of 8-bit×8-bit depth-wise conv. estimated from our benchmarking results on the cluster's cores, scaled considering that: due to improved ISA, our core is ∼10× faster on a per-core basis [25]; additional ∼7× improvement factor due to the cluster parallelism [28].

non-negligible area costs). In our architecture, weights can be stored directly on the non-volatile IMAs, without having to consider an external Flash.

The system presented in [61] consists of a PCM-based IMC array extended with digital logic that performs only activation and pooling operations, while a small SRAM memory acts as a layer-to-layer intermediate buffer. The higher peak performance and efficiency on MVMs they show compared to us is due to the bigger array size they used (1024×512 compared to 256×256), being the in-memory macro based on the same prototype [46], while a loss as little as 10% of the peak is attributable to the integration of the IMA into a complex system like the one we propose, as we show in Sec. 5.4.3. However, the architecture in [61] is too limited to execute the end-to-end inference of the MobileNetV2 for two main reasons: first, a single IMC array can not host all the layers

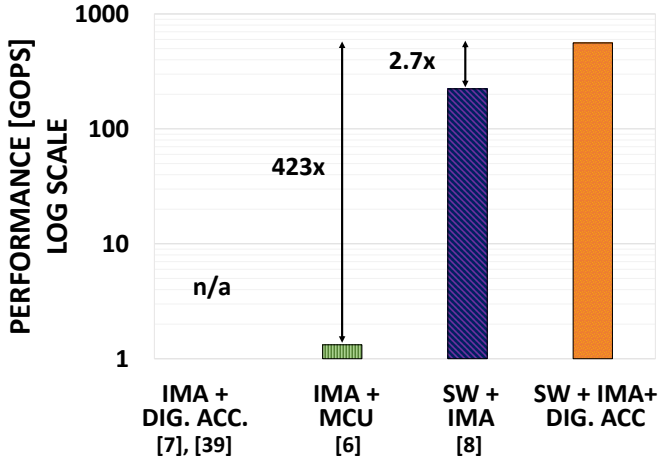


Figure 5.15: Performance of the MobileNetV2, on four IMC-based computing models. On the IMA+ASIC it is not possible to deploy the network model, due to architectural limitations.

weights of the MobileNet; second, there are no programmable cores to handle residual connections and control operations. Despite a more complex data-path compared to [61], including a cluster of 4×4 computing in memory units and a network-on-chip for communication which delivers outstanding performance and efficiency on MVMs, also the architecture shown in [62] is not viable to map heterogeneous workloads such as the MobileNetV2, due to the absence of a programmable processor.

Finally, to better highlight the contribution of this chapter, we abstract the specific System-on-Chip implementations described in Tab. 5.1 to four categories representative of the state-of-the-art, as shown in Fig. 5.15. We can highlight four different processing models: *i)* analog cores extended with fixed-function digital logic [61, 62] (IMA+ DIG. ACC.), *ii)* analog cores controlled by simple MCU-subsystems [94] (IMA+ MCU), *iii)* IMAs integrated into tightly-coupled clusters of programmable processors [95] (SW+ IMA), and *iv)* the paradigm proposed in this chapter, where we exploit heterogeneity both in terms of analog and digital computing and in terms of programmable cores and lightweight tightly coupled digital acceleration (SW+IMA+ DIG. ACC.).

Fig. 5.15 shows the results of the exploration, highlighting that for the MobileNetV2 workload, the computational model proposed in this chapter delivers significantly better performance compared to all the models exploiting programmable processor to sustain flexibility bottlenecks of IMC arrays. On the other hand, architectures only mixing specialized digital hardware with AIMC can only deal with DNN models for which they are designed, not being able to adapt to different models for which they were not intended before fabrication.

We argue that this concept, only demonstrated for MobileNetV2 DNN in Fig. 5.15 can be easily extended to more complex computer vision pipelines in the embedded

domain, where AI workloads are often coupled to more traditional linear algebra algorithms such as Principal Component Analysis (PCA), Fast Fourier Transform, Filtering Functions or Inverse Kinematics [118]. We believe that the approach proposed in this chapter is a viable way to tackle the performance and flexibility requirements of rapidly evolving modern computer vision pipelines.

Chapter 6

Conclusion

Empowering edge of the IoT devices with AI computing capabilities allows to distill the raw data acquired from the sensors in a much more dense format, extracting high-level information directly on the end-nodes. This new computing scenario provides several advantages, such as widening the IoT applications spectrum with AI-enhanced tasks, reducing the power for data transmission over wireless channels and eliminating security issues which are exacerbated by the enormously increasing factor of the interconnected IoT devices. This dissertation proposed, analyzed and compared different solutions to boost the computing capabilities of micro-controller class of devices, working at different levels of the hardware-software stack of the IoT environment.

The first contribution of this thesis was at the software level. Chapter 3 presented an optimized back-end library for QNN inference on top of tiny IoT devices, targeting the PULP platform. The library exploits the *XpulpV2* ISA and the cluster's parallelism of the underlying hardware and supports low-bitwidth integer computation with INT-8, INT-4, INT-2, and INT-1 data types. The software optimizations presented, tailored on the target hardware consisting of a computing cluster of 8 custom extended RISC-V processors, showed 63x performance improvements on convolution kernels over the same kernels implemented on a single processor featuring the RV32IMC ISA. The execution of a end-to-end INT-8 QNN on a commercial embodiment of the target platform, namely GAP8, outperformed by 19.49× the inference performance (in terms of cycles) of the network on an STM32H7 microcontroller, using the CMSIS-NN library. Furthermore, the energy efficiency achieved on a commercial embodiment of the PULP platform, namely GAP8, resulted to be 24 $GMAC/s/W$, 14.1× higher with respect to the highest efficient STM solution, the STM32L4 board; at the same time GAP8 achieved 1.066 $GMAC/s$, which is 7.45× higher than the performance of STM32H7 board, the high-end micro-controller system proposed by STM.

These results demonstrated a hardware-software solution that allows to achieve an energy proportionality on tiny micro-controllers where it is not needed to trade performance with energy efficiency, as opposed to commercial MCU solutions, enabling efficient and software programmable QNN inference on IoT class of devices. The proposed library has been also integrated as back-end library in a vertical software stack that comprises automatic data tiling (namely *Dory* [38]) and quantization-aware training (namely *Nemo* [119]) tools to fully handle the deployment and the end-to-end inference of real-sized neural network models such as the MobileNetV1 [80] on top of fully programmable IoT systems.

Despite the advancements with respect to state-of-the-art hardware-software solutions, Chapter 3 also highlighted the limits of an approach purely at the software level. The maximum performance and energy efficiency achievable are bounded by the hardware characteristics of the underlying platform. As detailed in Chapter 3, sub-byte and mixed-precision QNN kernels still suffer from drop-off in performance when compared to the symmetric 8-bit ones: the lack of support in the ISA of the target platform for sub-byte SIMD operations requires extra data manipulation operations that degrades the overall performance and efficiency.

In this scenario, the low-bitwidth integer arithmetic can only serve to compress the memory footprint of the DNN models, but not as a technique to improve the computing efficiency of the AI-enhanced applications running on IoT devices. Chapter 4 tackled this problem at the architectural and micro-architectural levels, by proposing a 2-bit to 16-bit multi-precision *Dotp* Unit, integrated into the RI5CY core (see Section 2.3 and Chapter 4), and optimized at the core level to guarantee high energy efficiency in dotp-based computation. Subsequently, to exploit the designed hardware, the core ISA has been extended with a set of low bit-width SIMD arithmetic instructions and related micro-architecture modifications to the datapath of the pipeline. Furthermore, the new core has been integrated in a multi-core computing cluster, showing a near-linear speedup of the performance compared to the single-core execution.

The implementation of the cluster with leading-edge GlobalFoundries 22nm 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. In perspective, the work presented in Chapter 4 showed at least two orders of magnitude improvements in performance and energy efficiency compared to existing state-of-the-art hardware and software solutions based on ARM Cortex-M cores. These achievements paves the way

to software programmable QNN inference at the extreme edge of the IoT, promising ASIC-like efficiency with way higher flexibility.

The contribution presented above brings the utilization of the core hardware resources up to 94%, close to the structural limit for the target processor. The performance bottleneck, in that case, moves towards the data communication at the interfaces between the core and the main memory. However, such scenario is common of traditional Von Neumann computing architectures, referred to as *Von Neumann bottleneck*. The emerging Analog in-memory computing (IMC) paradigm promises to overcome this limitation by processing the data within the memory boundaries and shows one to several orders of magnitude improvements in terms of energy efficiency, on MVM operations, compared to MCU and digital ASIC solutions, which is especially appealing for modern TinyML tasks running on battery powered IoT devices.

Nevertheless, IMC accelerators are outstanding platforms to deploy MVM based operations but they can not sustain the heterogeneity of the IoT workload. Hence, to target practical IoT applications IMC arrays must be enclosed in programmable heterogeneous systems, introducing new system-level challenges.

Chapter 5 aimed at exploring these challenges and at proposing a solution to empower IoT end-nodes with in-memory computing capabilities, by presenting a full implementation of a heterogeneous tightly-coupled clustered architecture integrating 8 RISC-V processors, a non-volatile PCM-based IMC accelerator, and a depth-wise digital accelerator, targeting the GlobalFoundries 22nm FD-SOI technology. The presented solution was benchmarked on a highly heterogeneous workload such as the *Bottleneck* layer, overcoming software execution of the layer by 11.5× and 9.5× in terms of performance and energy efficiency.

Furthermore, Chapter 5 presented a hardware design space exploration to investigate the system-level challenges and to show up the hardware resources necessary to enable end-to-end inference of real-world DNNs such as the *MobileNetV2*. The scaled-up system showed end-to-end inference execution 10× faster within 2.5× lower energy than fully digital solutions and more than two orders of magnitude faster than existing state-of-the-art analog/digital heterogeneous solutions. Such advancements demonstrated the effectiveness of the proposed heterogeneous computational model to sustain flexibility bottlenecks of IMC arrays. The concept presented here for the *MobileNetV2* can be easily extended to more complex computer vision pipelines in the embedded domain, where AI workloads are often coupled to more traditional linear algebra algorithms. It is to believe that the proposed approach is a viable solution to tackle the performance and flexibility requirements of rapidly evolving modern computer vision pipelines.

Appendix A

Abbreviations

AI Artificial Intelligence

AIMC Analog In-Memory Computing

ASIC Application Specific Integrated Circuit

CHW Channel Height Width

CNN Convolutional Neural Network

DL Deep Learning

DNN Deep Neural Network

DRAM Dynamic Random Access Memory

FD-SOI Fully Depleted Sylicon on Insulator

FPGA Field Programmable Gate Array

GP-GPU General Purpose- Graphic Processing Unit

HAL Hardware-Abstraction-Layer

HWC Height Width Channel

HWPE Hardware Processing Engine

IMA In-Memory Accelerator

IMC In-Memory Computing

IoT Internet of Things

ISA Instruction Set Architecture

MIMD Multiple-Instructions Multiple-Data

MCU Micro-Controller Units

ML Machine Learning

MRAM Magnetoresistive Random Access Memory

MVM Matrix-Vector Multiplication

NN Neural Network

NV Non-Volatile

PBBS Parallel Balanced-Bit-Serial

PCM Phase-Change Memory

PULP Parallel Ultra-Low Power

QNN Quantized Neural Network

ReRAM Resistive Random Access Memory

SIMD Single-Instruction Multiple-Data

SRAM Static Random Access Memory

SOT-MRAM Spin-orbit Torque MRAM

TinyML Tiny Machine Learning

Bibliography

[1] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P. Sheth. Machine learning for internet of things data analysis: a survey. *Digital Communications and Networks*, 4(3):161 – 175, 2018. ISSN 2352-8648. doi: https://doi.org/10.1016/j.dcan.2017.10.002.

[2] Naser Hossein Motlagh, Miloud Bagaa, and Tarik Taleb. UAV-based IoT platform: A crowd surveillance use case. *IEEE Communications Magazine*, 55(2):128–134, 2017.

[3] Olakunle Elijah, Tharek Abdul Rahman, Igbafe Orikumhi, Chee Yen Leow, and MHD Nour Hindia. An overview of Internet of Things (IoT) and data analytics in agriculture: Benefits and challenges. *IEEE Internet of Things Journal*, 5(5): 3758–3773, 2018.

[4] Moeen Hassanalieragh, Alex Page, Tolga Soyata, Gaurav Sharma, Mehmet Aktas, Gonzalo Mateos, Burak Kantarci, and Silvana 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*, pages 285–292. IEEE, 2015.

[5] Marcello Zanghieri, Simone Benatti, Alessio Burrello, Victor Kartsch, Francesco Conti, and Luca Benini. Robust Real-Time Embedded EMG Recognition Framework Using Temporal Convolutional Networks on a Multicore IoT Processor. *IEEE Transactions on Biomedical Circuits and Systems*, 2019.

[6] C Arcadius Tokognon, Bin Gao, Gui Yun Tian, and Yan Yan. Structural health monitoring framework based on Internet of Things: A survey. *IEEE Internet of Things Journal*, 4(3):619–635, 2017.

[7] Daniele Palossi, Antonio Loquercio, Francesco Conti, Eric Flamand, Davide Scaramuzza, and Luca Benini. A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones. *IEEE Internet of Things Journal*, 2019.

[8] Praveen Kumar Malik, Rohit Sharma, Rajesh Singh, Anita Gehlot, Suresh Chandra Satapathy, Waleed S Alnumay, Danilo Pelusi, Uttam Ghosh, and Janmenjoy Nayak. Industrial internet of things and its applications in industry 4.0: State of the art. *Computer Communications*, 166:125–139, 2021.

[9] Aradhana Behura. Intelligent automotive sector with iot (internet of things) and its consequential impact in vehicular ad hoc networks. In *Internet of Things and Its Applications*, pages 427–449. Springer, 2022.

[10] Diego GS Pivoto, Luiz FF de Almeida, Rodrigo da Rosa Righi, Joel JPC Rodrigues, Alexandre Baratella Lugli, and Antonio M Alberti. Cyber-physical systems architectures for industrial internet of things applications in industry 4.0: A literature review. *Journal of Manufacturing Systems*, 58:176–192, 2021.

[11] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[12] Hongxing Gao, Wei Tao, Dongchao Wen, Tse-Wei Chen, Kinya Osa, and Masami Kato. Ifq-net: Integrated fixed-point quantization networks for embedded vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 607–615, 2018.

[13] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[14] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8612–8620, 2019.

[15] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry 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*, pages 2704–2713, 2018.

[16] Manuele Rusci, Alessandro Capotondi, and Luca Benini. Memory-Driven Mixed Low Precision Quantization For Enabling Deep Network Inference On Microcontrollers. *arXiv preprint arXiv:1905.13082*, 2019.

[17] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Survey and benchmarking of machine learning accelerators. *arXiv preprint arXiv:1908.11348*, 2019.

[18] Denis C Daly, Laura C Fujino, and Kenneth C Smith. Through the Looking Glass-2020 Edition: Trends in Solid-State Circuits From ISSCC. *IEEE Solid-State Circuits Magazine*, 12(1):8–24, 2020.

[19] Arm.project trillium machine learning platform. `https://www.arm.com/products/silicon-ip-cpu/machine-learning/project-trillium`, 2019.

[20] Eric Flamand, Davide Rossi, Francesco Conti, Igor Loi, Antonio Pullini, Florent Rotenberg, and Luca 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)*, pages 1–4. IEEE, 2018.

[21] 2018. Kendryte: K210 datasheet. `https://s3.cn-north-1.amazonaws.com.cn/dl.kendryte.com/documents/kendryte_datasheet_20181011163248_en.pdf`, 2018.

[22] Liangzhen Lai, Naveen Suda, and Vikas Chandra. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv preprint arXiv:1801.06601*, 2018.

[23] Manuele Rusci, Alessandro Capotondi, Francesco Conti, and Luca 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)*, pages 1–2. IEEE, 2018.

[24] Antonio Pullini, Davide Rossi, Igor Loi, Giuseppe Tagliavini, and Luca Benini. Mr. wolf: An energy-precision scalable parallel ultra low power soc for iot edge processing. *IEEE Journal of Solid-State Circuits*, 54(7):1970–1981, 2019.

[25] Michael Gautschi, Pasquale Davide Schiavone, Andreas Traber, Igor Loi, Antonio Pullini, Davide Rossi, Eric Flamand, Frank K Gürkaynak, and Luca Benini. Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2700–2713, 2017.

[26] Distinguished Engineer Joseph Yiu. Introduction to the Arm Cortex-M55 Processor. Available online: https://pages.arm.com/cortex-m55-introduction.html, February 2020.

[27] Angelo Garofalo, Manuele Rusci, Francesco Conti, Davide Rossi, and Luca Benini. Pulp-nn: A computing library for quantized neural network inference at the edge on risc-v based parallel ultra low power clusters. In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 33–36. IEEE, 2019.

[28] Angelo Garofalo, Manuele Rusci, Francesco Conti, Davide Rossi, and Luca Benini. PULP-NN: accelerating quantized neural networks on parallel ultra-low-power RISC-V processors. *Philosophical Transactions of the Royal Society A*, 378(2164): 20190155, 2020.

[29] Nazareno Bruschi, Angelo Garofalo, Francesco Conti, Giuseppe Tagliavini, and Davide Rossi. Enabling mixed-precision quantized neural networks in extreme-edge devices. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*, pages 217–220, 2020.

[30] STMicroelectronics. 2018. STM32L476 datasheet. `https://www.st.com/resource/en/datasheet/stm32l476je.pdf`, 2018.

[31] STMicroelectronics. 2018. STM32H743 datasheet. `https://www.st.com/resource/en/datasheet/stm32h743bi.pdf`, 2018.

[32] Angelo Garofalo, Giuseppe Tagliavini, Francesco Conti, Davide Rossi, and Luca Benini. Xpulpnn: accelerating quantized neural networks on risc-v processors through isa extensions. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 186–191. IEEE, 2020.

[33] Angelo Garofalo, Giuseppe Tagliavini, Francesco Conti, Luca Benini, and Davide Rossi. XpulpNN: Enabling Energy Efficient and Flexible Inference of Quantized Neural Networks on RISC-V based IoT End Nodes. *IEEE Transactions on Emerging Topics in Computing*, 2021.

[34] Angelo Garofalo, Gianmarco Ottavi, Alfio di Mauro, Francesco Conti, Giuseppe Tagliavini, Luca Benini, and Davide Rossi. A 1.15 TOPS/W, 16-Cores Parallel Ultra-Low Power Cluster with 2b-to-32b Fully Flexible Bit-Precision and Vector Lockstep Execution Mode. In *ESSCIRC 2021-IEEE 47th European Solid State Circuits Conference (ESSCIRC)*, pages 267–270. IEEE, 2021.

[35] Angelo Garofalo, Gianmarco Ottavi, Francesco Conti, Geethan Karunaratne, Irem Boybat, Luca Benini, and Davide Rossi. A Heterogeneous In-Memory Computing Cluster For Flexible End-to-End Inference of Real-World Deep Neural Networks. *arXiv*, 2022.

[36] Alessio Burrello, Francesco Conti, Angelo Garofalo, Davide Rossi, and Luca Benini. Work-in-progress: Dory: lightweight memory hierarchy management for deep nn inference on iot endnodes. In *2019 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 1–2. IEEE, 2019.

[37] Gianmarco Ottavi, Angelo Garofalo, Giuseppe Tagliavini, Francesco Conti, Luca Benini, and Davide Rossi. A mixed-precision RISC-V processor for extreme-edge DNN inference. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 512–517. IEEE, 2020.

[38] Alessio Burrello, Angelo Garofalo, Nazareno Bruschi, Giuseppe Tagliavini, Davide Rossi, and Francesco Conti. Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus. *IEEE Transactions on Computers*, 2021.

[39] Annachiara Ruospo, Riccardo Cantoro, Ernesto Sanchez, Pasquale Davide Schiavone, Angelo Garofalo, and Luca Benini. On-line Testing for Autonomous Systems driven by RISC-V Processor Design Verification. In *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2019.

[40] Fabio Montagna, Stefan Mach, Simone Benatti, Angelo Garofalo, Gianmarco Ottavi, Luca Benini, Davide Rossi, and Giuseppe Tagliavini. A Low-Power Transprecision Floating-Point Cluster for Efficient Near-Sensor Data Analytics. *IEEE Transactions on Parallel and Distributed Systems*, 33(5):1038–1053, 2021.

[41] Fabio Montagna, Giuseppe Tagliavini, Davide Rossi, Angelo Garofalo, and Luca Benini. Streamlining the OpenMP Programming Model on Ultra-Low-Power Multi-core MCUs. In *International Conference on Architecture of Computing Systems*, pages 167–182. Springer, 2021.

[42] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[43] Bert Moons, Koen Goetschalckx, Nick Van Berckelaer, and Marian Verhelst. Minimum energy quantized neural networks. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pages 1921–1925. IEEE, 2017.

[44] Francesco Conti and Luca Benini. A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 683–688. EDA Consortium, 2015.

[45] Andrew Waterman, Yunsup Lee, David A Patterson, and Krste Asanović. The risc-v instruction set manual, volume i: User-level isa, version 2.1. *RISC-V*, 2016.

[46] R Khaddam-Aljameh, M Stanisavljevic, J Fornt Mas, G Karunaratne, M Braendli, F Liu, A Singh, SM Müller, U Egger, A Petropoulos, et al. HERMES Core–A 14nm

CMOS and PCM-based In-Memory Compute Core using an array of 300ps/LSB Linearized CCO-based ADCs and local digital processing. In *2021 Symposium on VLSI Circuits*, pages 1–2. IEEE, 2021.

[47] Vinay Joshi, Manuel Le Gallo, Simon Haefeli, Irem Boybat, Sasidharan Rajalekshmi Nandakumar, Christophe Piveteau, Martino Dazzi, Bipin Rajendran, Abu Sebastian, and Evangelos Eleftheriou. Accurate deep neural network inference using computational phase-change memory. *Nature communications*, 11(2473), 2020.

[48] Davide Rossi, Antonio Pullini, Igor Loi, Michael Gautschi, Frank Kağan Gürkaynak, Adam Teman, Jeremy Constantin, Andreas Burg, Ivan Miro-Panades, Edith Beignè, et al. Energy-efficient near-threshold parallel computing: The pulpv2 cluster. *Ieee Micro*, 37(5):20–31, 2017.

[49] Giuseppe Desoli, Nitin Chawla, Thomas Boesch, Surinder-pal Singh, Elio Guidetti, Fabio De Ambroggi, Tommaso Majo, Paolo Zambotti, Manuj Ayodhyawasi, Harvinder 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)*, pages 238–239. IEEE, 2017.

[50] Lukas Cavigelli and Luca Benini. Origami: A 803-gop/s/w convolutional network accelerator. *IEEE Transactions on Circuits and Systems for Video Technology*, 27 (11):2461–2475, 2017.

[51] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[52] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[53] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[54] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[55] Jinmook Lee, Changhyeon Kim, Sanghoon Kang, Dongjoo Shin, Sangyeob Kim, and Hoi-Jun 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)*, pages 218–220. IEEE, 2018.

[56] Bert Moons, Roel Uytterhoeven, Wim Dehaene, and Marian 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)*, pages 246–247. IEEE, 2017.

[57] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. YodaNN: An architecture for ultralow power binary-weight CNN acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(1):48–60, 2018.

[58] Kota Ando, Kodai Ueyoshi, Kentaro Orimo, Haruyoshi Yonekawa, Shimpei Sato, Hiroki Nakahara, Shinya Takamaeda-Yamazaki, Masayuki Ikebe, Tetsuya Asai, Tadahiro Kuroda, et al. BRein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W. *IEEE Journal of Solid-State Circuits*, 53(4):983–994, 2018.

[59] Avishek Biswas and Anantha P Chandrakasan. CONV-SRAM: An energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 54(1):217–230, 2018.

[60] Naveen Verma, Hongyang Jia, Hossein Valavi, Yinqi Tang, Murat Ozatay, Lung-Yen Chen, Bonan Zhang, and Peter Deaville. In-memory computing: Advances and prospects. *IEEE Solid-State Circuits Magazine*, 11(3):43–55, 2019.

[61] Chuteng Zhou, Fernando Garcia Redondo, Julian Büchel, Irem Boybat, Xavier Timoneda Comas, S. R. Nandakumar, Shidhartha Das, Abu Sebastian, Manuel Le Gallo, and Paul N. Whatmough. AnalogNets: ML-HW Co-Design of Noise-robust TinyML Models and Always-On Analog Compute-in-Memory Accelerator, 2021.

[62] Hongyang Jia, Murat Ozatay, Yinqi Tang, Hossein Valavi, Rakshit Pathak, Jinseok Lee, and Naveen Verma. Scalable and Programmable Neural Network Inference Accelerator Based on In-Memory Computing. *IEEE Journal of Solid-State Circuits*, pages 1–1, 2021. doi: 10.1109/JSSC.2021.3119018.

[63] Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. Memory devices and applications for in-memory computing. *Nature nanotechnology*, 15(7):529–544, 2020.

[64] Vinayak Gokhale, Aliasger Zaidy, Andre Xian Ming Chang, and Eugenio Culurciello. Snowflake: An efficient hardware accelerator for convolutional neural networks. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017.

[65] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun 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)*, pages 1–8. IEEE, 2017.

[66] Stylianos I Venieris and Christos-Savvas Bouganis. Latency-driven design for FPGA-based convolutional neural networks. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE, 2017.

[67] Paolo Meloni, Alessandro Capotondi, Gianfranco Deriu, Michele Brian, Francesco Conti, Davide Rossi, Luigi Raffo, and Luca Benini. NEURA ghe: Exploiting CPU-FPGA Synergies for Efficient and Flexible CNN Inference Acceleration on Zynq SoCs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 11(3):18, 2018.

[68] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen 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*, pages 26–35. ACM, 2016.

[69] Adrien Prost-Boucle, Alban Bourge, Frédéric Pétrot, Hande Alemdar, Nicholas Caldwell, and Vincent Leroy. Scalable high-performance architecture for convolutional ternary neural networks on FPGA. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–7. IEEE, 2017.

[70] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees 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*, pages 65–74. ACM, 2017.

[71] Lattice. 2019. Lattice sensAI Delivers 10X Performance Boost for Low Power, Smart IoT Devices at the Edge. `https://www.latticesemi.com/About/Newsroom/PressReleases/2019/201911sensAI`, 2019.

[72] Nvidia. 2015. NVIDIA Tegra X1. `https://international.download.nvidia.com/pdf/tegra/Tegra-X1-whitepaper-v1.0.pdf`, 2015.

[73] Nvidia. 2015. GPU-Based Deep Learning Inference: A Performance and Power Analysis. `https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf`, 2015.

[74] Nvidia. 2018, September. NVIDIA Turing Architecture In-Depth. https://devblogs.nvidia.com/nvidia-turing-architecture-in-depth/, 2018.

[75] Raspberry Pi Compute Module 3+. 2019. https://www.raspberrypi.org/documentation/hardware/computemodule/datasheets/rpi_DATA_CM3plus_1p0.pdf, 2018.

[76] Francesco Conti, Robert Schilling, Pasquale Davide Schiavone, Antonio Pullini, Davide Rossi, Frank Kağan Gürkaynak, Michael Muehlberghuber, Michael Gautschi, Igor Loi, Germain Haugou, et al. An IoT endpoint system-on-chip for secure and energy-efficient near-sensor analytics. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(9):2481–2494, 2017.

[77] ARM. 2019. Armv8.1-M architecture. https://pages.arm.com/introduction-armv8.1m.html?_ga=2.237285124.508798244.1553788782-2017191492.1542023072, 2019.

[78] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

[79] Igor Loi, Alessandro Capotondi, Davide Rossi, Andrea Marongiu, and Luca Benini. The quest for energy-efficient I $ design in ultra-low-power clustered many-cores. *IEEE Transactions on Multi-Scale Computing Systems*, 4(2):99–112, 2018.

[80] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[81] Alessandro Capotondi, Manuele Rusci, Marco Fariselli, and Luca Benini. Cmix-nn: Mixed low-precision cnn library for memory-constrained edge devices. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(5):871–875, 2020.

[82] Francesco Conti. Technical Report: NEMO DNN Quantization for Deployment Model, 2020.

[83] STMicroelectronics. 2018. X-CUBE-AI (data brief). Artificial intelligence (AI) software expansion for STM32Cube. https://www.st.com/resource/en/data_brief/x-cube-ai.pdf, 2018.

[84] Bing-Chen Wu and I-Chyn 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*, 65(1):141–153, 2017.

[85] Arm. MICRONPU ETHOS-U55. Available online: https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u55, 2020.

[86] 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, 2020.

[87] D. Bol, M. Schramme, L. Moreau, P. Xu, R. Dekimpe, R. Saeidi, T. Haine, C. Frenkel, and D. Flandre. SleepRunner: A 28-nm FDSOI ULP Cortex-M0 MCU With ULL SRAM and UFBR PVT Compensation for 2.6-3.6-μW/DMIPS 40-80-MHz Active Mode and 131-nW/kB Fully Retentive Deep-Sleep Mode. *IEEE Journal of Solid-State Circuits*, pages 1–1, 2021. doi: 10.1109/JSSC.2021.3056219.

[88] I. Miro-Panades, B. Tain, J. F. Christmann, D. Coriat, R. Lemaire, C. Jany, B. Martineau, F. Chaix, A. Quelen, E. Pluchart, J. P. Noel, R. Boumchedda, A. Makosiej, M. Montoya, S. Bacles-Min, D. Briand, J. M. Philippe, A. Valentian, F. Heitzmann, E. Beigne, and F. Clermidy. SamurAI: A 1.7MOPS-36GOPS Adaptive Versatile IoT Node with 15,000× Peak-to-Idle Power Reduction, 207ns Wake-Up Time and 1.3TOPS/W ML Efficiency. In *2020 IEEE Symposium on VLSI Circuits*, pages 1–2, 2020. doi: 10.1109/VLSICircuits18222.2020.9163000.

[89] Davide Rossi, Francesco Conti, Manuel Eggimann, Alfio Di Mauro, Giuseppe Tagliavini, Stefan Mach, Marco Guermandi, Antonio Pullini, Igor Loi, Jie Chen, et al. Vega: A Ten-Core SoC for IoT Endnodes With DNN Acceleration and Cognitive Wake-Up From MRAM-Based State-Retentive Sleep Mode. *IEEE Journal of Solid-State Circuits*, 2021.

[90] Shimeng Yu, Xiaoyu Sun, Xiaochen Peng, and Shanshi Huang. Compute-in-memory with emerging nonvolatile-memories: Challenges and prospects. In *2020 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4. IEEE, 2020.

[91] Artificial intelligence for a safer, greener and more trusted world. `https://www.axelera.ai/`, 2021. Accessed: 2021-12-14.

[92] Dave Fick and Mike Henry. Analog Computation in Flash Memory for Datacenter-scale AI Inference in a Small Chip. In *Hot Chips*, 2018.

[93] SR Nandakumar, Manuel Le Gallo, Christophe Piveteau, Vinay Joshi, Giovanni Mariani, Irem Boybat, Geethan Karunaratne, Riduan Khaddam-Aljameh, Urs Egger, Anastasios Petropoulos, et al. Mixed-precision deep learning based on computational memory. *Frontiers in neuroscience*, 14:406, 2020.

[94] Hongyang Jia, Hossein Valavi, Yinqi Tang, Jintao Zhang, and Naveen Verma. A programmable heterogeneous microprocessor based on bit-scalable in-memory computing. *IEEE Journal of Solid-State Circuits*, 55(9):2609–2621, 2020.

[95] Gianmarco Ottavi, Geethan Karunaratne, Francesco Conti, Irem Boybat, Luca Benini, and Davide Rossi. End-to-end 100-TOPS/W Inference With Analog In-Memory Computing: Are We There Yet? In *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–4. IEEE, 2021.

[96] Daniele Ielmini and H-S Philip Wong. In-memory computing with resistive switching devices. *Nature Electronics*, 1(6):333–343, 2018.

[97] Wei-Hao Chen, Kai-Xiang Li, Wei-Yu Lin, Kuo-Hsiang Hsu, Pin-Yi Li, Cheng-Han Yang, Cheng-Xin Xue, En-Yu Yang, Yen-Kai Chen, Yun-Sheng Chang, Tzu-Hsiang Hsu, Ya-Chin King, Chorng-Jung Lin, Ren-Shuo Liu, Chih-Cheng Hsieh, Kea-Tiong Tang, and Meng-Fan Chang. A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pages 494–496, 2018. doi: 10.1109/ISSCC.2018.8310400.

[98] J Doevenspeck, Kevin Garello, B Verhoef, R Degraeve, S Van Beek, D Crotti, F Yasin, S Couet, G Jayakumar, IA Papistas, et al. SOT-MRAM based analog in-memory computing for DNN inference. In *2020 IEEE Symposium on VLSI Technology*, pages 1–2. IEEE, 2020.

[99] Vinay Joshi, Manuel Le Gallo, Simon Haefeli, Irem Boybat, Sasidharan Rajalekshmi Nandakumar, Christophe Piveteau, Martino Dazzi, Bipin Rajendran, Abu Sebastian, and Evangelos Eleftheriou. Accurate deep neural network inference using computational phase-change memory. *Nature communications*, 11(1):1–13, 2020.

[100] Yu-Der Chih, Po-Hao Lee, Hidehiro Fujiwara, Yi-Chun Shih, Chia-Fu Lee, Rawan Naous, Yu-Lin Chen, Chieh-Pu Lo, Cheng-Han Lu, Haruki Mori, et al. An 89top-s/w and 16.3 tops/mm 2 all-digital sram-based full-precision compute-in memory macro in 22nm for machine-learning edge applications. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 252–254. IEEE, 2021.

[101] Hyunjoon Kim, Qian Chen, Taegeun Yoo, Tony Tae-Hyoung Kim, and Bongjin Kim. A 1-16b precision reconfigurable digital in-memory computing macro featuring column-mac architecture and bit-serial computation. In *ESSCIRC 2019-IEEE*

45th European Solid State Circuits Conference (ESSCIRC), pages 345–348. IEEE, 2019.

[102] Sparsh Mittal, Gaurav Verma, Brajesh Kaushik, and Farooq A Khanday. A survey of SRAM-based in-memory computing techniques and applications. *Journal of Systems Architecture*, 119:102276, 2021.

[103] Shihui Yin, Zhewei Jiang, Jae-Sun Seo, and Mingoo Seok. XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks. *IEEE Journal of Solid-State Circuits*, 55(6):1733–1743, 2020.

[104] Jinshan Yue, Xiaoyu Feng, Yifan He, Yuxuan Huang, Yipeng Wang, Zhe Yuan, Mingtao Zhan, Jiaxin Liu, Jian-Wei Su, Yen-Lin Chung, et al. A 2.75-to-75.9 TOP-S/W computing-in-memory NN processor supporting set-associate block-wise zero skipping and ping-pong CIM with simultaneous computation and weight updating. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 238–240. IEEE, 2021.

[105] Jinseok Lee, Hossein Valavi, Yinqi Tang, and Naveen Verma. Fully Row/Column-Parallel In-memory Computing SRAM Macro employing Capacitor-based Mixed-signal Computation with 5-b Inputs. In *2021 Symposium on VLSI Circuits*, pages 1–2. IEEE, 2021.

[106] Kaushik Roy, Indranil Chakraborty, Mustafa Ali, Aayush Ankit, and Amogh Agrawal. In-memory computing in emerging memory technologies for machine learning: an overview. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[107] Abu Sebastian, Irem Boybat, Martino Dazzi, Iason Giannopoulos, V Jonnalagadda, Vinay Joshi, Geethan Karunaratne, Benedikt Kersting, Riduan Khaddam-Aljameh, SR Nandakumar, et al. Computational memory-based inference and training of deep neural networks. In *2019 Symposium on VLSI Technology*, pages T168–T169. IEEE, 2019.

[108] Cheng-Xin Xue, Tsung-Yuan Huang, Je-Syu Liu, Ting-Wei Chang, Hui-Yao Kao, Jing-Hong Wang, Ta-Wei Liu, Shih-Ying Wei, Sheng-Po Huang, Wei-Chen Wei, et al. 15.4 a 22nm 2mb reram compute-in-memory macro with 121-28tops/w for multibit mac computing for tiny ai edge devices. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 244–246. IEEE, 2020.

[109] Cheng-Xin Xue, Je-Min Hung, Hui-Yao Kao, Yen-Hsiang Huang, Sheng-Po Huang, Fu-Chun Chang, Peng Chen, Ta-Wei Liu, Chuan-Jia Jhang, Chin-I Su, et al. A 22nm 4mb 8b-precision reram computing-in-memory macro with 11.91 to 195.7

tops/w for tiny ai edge devices. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 245–247. IEEE, 2021.

[110] Manuel Le Gallo, Abu Sebastian, Roland Mathis, Matteo Manica, Heiner Giefers, Tomas Tuma, Costas Bekas, Alessandro Curioni, and Evangelos Eleftheriou. Mixed-precision in-memory computing. *Nature Electronics*, 1(4):246–253, 2018.

[111] Daniele Palossi, Nicky Zimmerman, Alessio Burrello, Francesco Conti, Hanna Müller, Luca Maria Gambardella, Luca Benini, Alessandro Giusti, and Jérôme Guzzi. Fully Onboard AI-powered Human-Drone Pose Estimation on Ultra-low Power Autonomous Flying Nano-UAVs. *IEEE Internet of Things Journal*, pages 1–1, 2021. doi: 10.1109/JIOT.2021.3091643.

[112] Martino Dazzi, Abu Sebastian, Thomas Parnell, Pier Andrea Francese, Luca Benini, and Evangelos Eleftheriou. Efficient pipelined execution of CNNs based on in-memory computing and graph homomorphism verification. *IEEE Transactions on Computers*, 70(6):922–935, 2021.

[113] Pouya Houshmand, Stefan Cosemans, Linyan Mei, Ioannis Papistas, Debjyoti Bhattacharjee, Peter Debacker, Arindam Mallik, Diederik Verkest, and Marian Verhelst. Opportunities and limitations of emerging analog in-memory compute dnn architectures. In *2020 IEEE International Electron Devices Meeting (IEDM)*, pages 29–1. IEEE, 2020.

[114] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[115] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[116] P Narayanan, S Ambrogio, A Okazaki, K Hosokawa, H Tsai, A Nomura, T Yasuda, C Mackin, SC Lewis, A Friz, et al. Fully on-chip MAC at 14nm enabled by accurate row-wise programming of PCM-based weights and parallel vector-transport in duration-format. In *2021 Symposium on VLSI Technology*, pages 1–2. IEEE, 2021.

[117] Jukka Jylänki. A thousand ways to pack the bin-a practical approach to two-dimensional rectangle bin packing. *retrived from http://clb. demon. fi/files/RectangleBinPack. pdf*, 2010.

[118] Andreas Aristidou, Joan Lasenby, Yiorgos Chrysanthou, and Ariel Shamir. Inverse kinematics techniques in computer graphics: A survey. In *Computer Graphics Forum*, volume 37. Wiley Online Library, 2018.

[119] Francesco Conti. Technical Report: NEMO DNN Quantization for Deployment Model. *arXiv preprint arXiv:2004.05930*, 2020.

[120] Vivienne Sze, Yu-Hsin Chen, Joel Emer, Amr Suleiman, and Zhengdong Zhang. Hardware for machine learning: Challenges and opportunities. In *2017 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–8. IEEE, 2017.

[121] Lattice. 2018. Accelerating Implementation of Low Power Artificial intelligence at the edge. `http://www.latticesemi.com/view_document?document_id=52384`, 2018.

[122] Tensor Flow Lite.2018. `https://www.tensorflow.org/lite/`, 2018.

[123] Shancang Li, Li Da Xu, and Shanshan Zhao. 5G Internet of Things: A survey. *Journal of Industrial Information Integration*, 10:1–9, 2018.

[124] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-Aware Automated Quantization. *arXiv preprint arXiv:1811.08886*, 2018.