A Heterogeneous In-Memory Computing Cluster for Flexible End-to-End Inference of Real-World Deep Neural Networks

A Heterogeneous In-Memory Computing Cluster For Flexible End-to-End Inference of Real-World Deep Neural Networks

Angelo Garofalo*, Gianmarco Ottavi*, Francesco Conti*,
Geethan Karunaratne†‡, Irem Boybat†, Luca Benini‡*, Davide Rossi*
*University of Bologna, Bologna, Italy †IBM Research Europe, Zurich, Switzerland ‡ETH Zurich, Zurich, Switzerland
{angelo.garofalo, gianmarco.ottavi2, davide.rossi, f.conti}@unibo.it {lbenini}@iis.ee.ethz.ch {kar, ibo}@zurich.ibm.com

***Abstract*—Deployment of modern TinyML tasks on small battery-constrained IoT devices requires high computational energy efficiency. Analog In-Memory Computing (IMC) using non-volatile memory (NVM) promises major efficiency improvements in deep neural network (DNN) inference and serves as on-chip memory storage for DNN weights. However, IMC's functional flexibility limitations and their impact on performance, energy, and area efficiency are not yet fully understood at the system level. To target practical end-to-end IoT applications, IMC arrays must be enclosed in heterogeneous programmable systems, introducing new system-level challenges which we aim at addressing in this work. We present a heterogeneous tightly-coupled clustered architecture integrating 8 RISC-V cores, an in-memory computing accelerator (IMA), and digital accelerators. We benchmark the system on a highly heterogeneous workload such as the Bottleneck layer from a MobileNetV2, showing 11.5× performance and 9.5× energy efficiency improvements, compared to highly optimized parallel execution on the cores. Furthermore, we explore the requirements for end-to-end inference of a full mobile-grade DNN (MobileNetV2) in terms of IMC array resources, by scaling up our heterogeneous architecture to a multi-array accelerator. Our results show that our solution, on the end-to-end inference of the 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.**

***Index Terms*—In-memory computing, RISC-V, Heterogeneous computing architecture, MobileNetV2**

I. Introduction

Analog in-memory computing (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 [1]. Both charge-based memory technologies (e.g. SRAM, DRAM, and flash) and resistance-based memory technologies (e.g. RRAM, PCM, and STT-MRAM) can serve as elements for such computational units [2].

This work was supported by the EU Horizon 2020 Research and Innovation projects WiPLASH (g.a. no. 863337) and European Pilot (g.a. 101034126).

Several demonstrations of AIMC-based architectures have appeared in the field of Deep Neural Network (DNN) inference acceleration, showing outstanding peak energy efficiency in the order of hundreds of TOPS/W [1], [2]. Industry interest in this technology is growing [3], [4]. 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 [1], [2].

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 [2]; difficulties in dealing with low-precision computations that are often the only ones supported by AIMC-based architectures [2]; the necessity of specialized training [5]. 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 [6] or specialized digital accelerators [7] 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 [6]. 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 [1]. 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 [6].

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 [6], will likely hit Amdahl's effect caused by the single-core bottleneck, hindering the whole computation performance.

In this work, we address the system-level challenges of analog IMC by exploiting extreme heterogeneity. The main contributions of this paper are the following:

- We design 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; we present a post place&route silicon-ready implementation targeting GlobalFoundries 22nm FDX technology;
- We optimize 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 [6];
- We benchmark the system 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 $2.6\times$ better performance and $2.8\times$ better energy efficiency compared to executing the layers on our previous work that integrates only 8 programmable cores and the IMC analog array [8];
- We scale up the proposed system to analyze the challenges and the hardware resources necessary to enable end-to-end inference of a MobileNetV2. Our architectural paradigm executes inference in $10ms$ with an energy of $482\mu J$, improving upon fully digital state-of-the-art solutions (SoA) [9] by $10\times$ in latency, reducing the energy consumption by $2.5\times$. Compared to SoA analog/digital architectures [6], our solution shows two orders of magnitude improvements in terms of execution latency.

The manuscript is organized as follows: in Sec. II we review the state-of-the-art; in Sec. III we outline the background, and in Sec. IV we present the heterogeneous cluster architecture. Then, in Sec. V and Sec. VI we report the experimental results, discussions, and explorations. Finally in Sec. VII we compare with state-of-the-art solutions. Sec. VIII concludes the paper.

II. Related Work

Charge-based memory technologies (e.g. SRAM [10], DRAM, Flash) and non-volatile (NV) resistive memory technologies [11] (e.g. ReRAM [12] PCM [2] and MRAM [13]) 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.

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 [1], [2]. 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 [14]. Also, noise-robust networks are an active research field for IMC deployment [15].

2) SRAM technology: The 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 [16]. In the analog domain, SRAMs can map MVMs by exploiting capacitive charge redistribution mechanisms along the bit-lines of the memory array [2]. 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 [17].

Most SoA academic SRAM-based IMC arrays operate in the analog domain [18]. One of the first prototypes appeared in 2018 [10], targeting binary-weight Neural Networks and demonstrating top-1 accuracy comparable with software accuracy on the MNIST dataset ($\sim$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 [19], as well as for reconfigurable bit-precision MVM operations showing80 TOPS/W [20]. Other SRAM-based IMC architectures have been proposed, achieving similar accuracy and efficiency [21]. 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 [2], and non-linearities [1].

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 [22]. Moreover, resistive memories such as Resistive RAM (ReRAM), magnetic RAM (MRAM), and phase-change memory (PCM), show other important advantages: non-volatility (NV), low power envelope, and multi-level storage [12]. 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) [23].

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.* [12] demonstrate significant computing parallelism, performing 8k MAC operations simultaneously. Other works show ReRAM-based IMC arrays as dense as 2Mb [24] or 4Mb [25], 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 [24], 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.* [13] present a 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 [26] and in DNN inference and even training tasks [23], showing limited error in the computation and much higher efficiency compared to traditional approaches [2].

Khaddam-Aljameh *et al.* [27] 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 [27].

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 [1], 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 [28]. 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 [1].

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.* [29] 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.* [30] 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.* [7] 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.* [31] propose a 4×4 array of cores consisting of charge-based IMC cross-bars 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 [6] 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 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

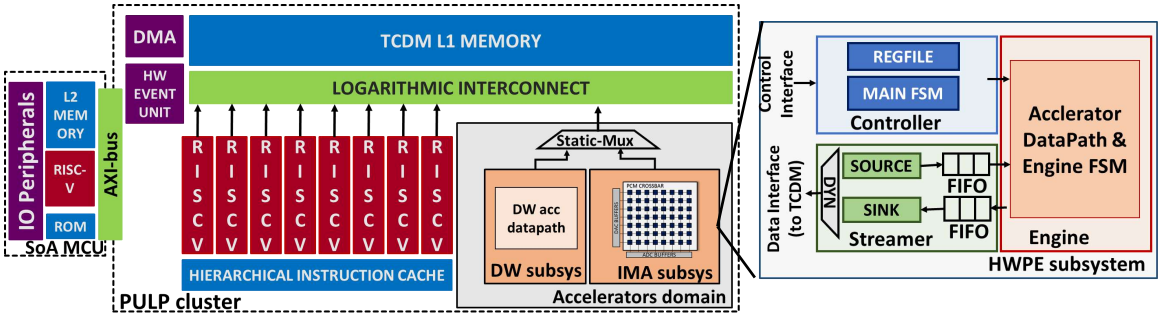


Fig. 1. Overview of the PULP cluster architecture, integrating the IMA and the digital depth-wise accelerator. Each accelerator is enclosed into a Hardware Processing Engine (HWPE) subsystem, depicted on the right.

functions with sufficient performance level.

The limits of the above-mentioned systems are mainly two: the integration of the IMC accelerator (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.

In this manuscript, we extend the work we presented in [8] by coupling the IMA with a novel design of a digital accelerator to improve the efficiency of depth-wise kernels, integrated into the heterogeneous system and we fully implement the cluster in the GlobalFoundries 22nm FDX technology. Furthermore, we scale up the architecture and 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.

III. Background

A. PCM-based In-Memory Accelerator

In this paper, we use the SoA In-Memory Computing array presented in [27] 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 cross-points 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). 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 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 $\mathbf{A}$ 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 [32]. 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 [33]. In the IMC cross-bar, 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.

B. The PULP Cluster

Highly parallelizable workloads such as DNNs are a good fit for high core count heterogeneous systems that can integrate specialized accelerators. The PULP cluster [9] we assume as a reference, depicted in Fig. 1, incorporates 8 RISC-V cores, each featuring a 4-stage in-order single-issue pipeline and implementing the RISC-V RV32IMCXpulpV2 Instruction Set Architecture (ISA). XpulpV2 is a custom extension to the RISC-V ISA [34] meant to accelerate arithmetic intensive kernels.

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, in this context, is 512 kB, divided on 32 banks. 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 offloaded to accelerators, integrated into the cluster through a standardized interface [35], discussed in Sec. IV-A.

The cluster communicates with a micro-controller system 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.

IV. Heterogeneous System

In this section, we present the analog in-memory accelerator (IMA), the depth-wise digital accelerator, and their integration into the PULP cluster through a standardized interface called Hardware Processing Engine.

A. 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.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, we expose to the programmer a set of hardware-abstraction-layer (HAL) functions that 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 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.

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

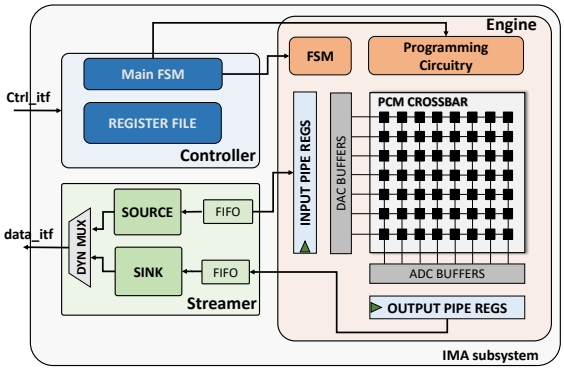


Fig. 2. IMA enclosed into the HWPE interface. The *data_itf* width is designed to match the IO requirements of the IMA.

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.

B. IMA Subsystem Architecture

Fig. 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 core, as shown in Sec. V-B. 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. 3

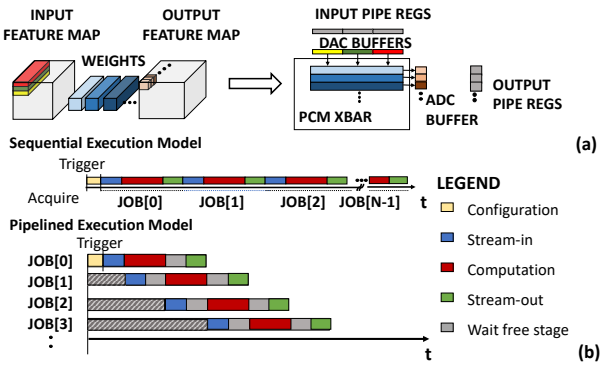


Fig. 3. (a) Mapping of standard convolutions on the PCM crossbar. (b) Timeline of the sequential and pipelined execution models.

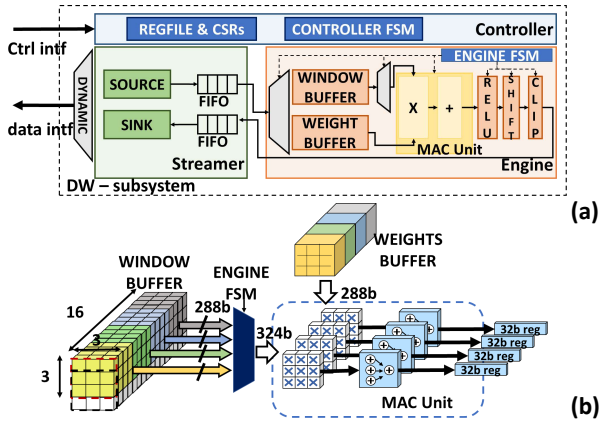


Fig. 4. (a) Architecture overview of the Depth-wise digital accelerator, enclosed in the HWPE. (b) Execution flow of the depth-wise operation.

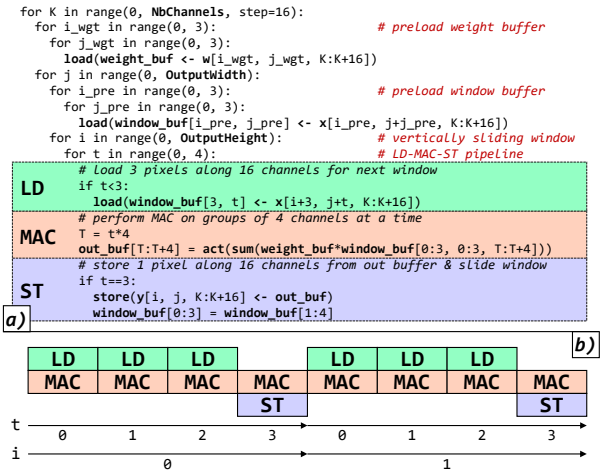


Fig. 5. (a) Pseudo-Python code describing the operation of the depth-wise accelerator datapath. (b) Detail of the *LD - MAC - ST* pipeline.

shows how a CNN layer is mapped on the IMA array. For a standard convolution, the streamers can directly perform a virtual IM2COL transformation [36], enabling to remap the computation to matrix-vector products of the form discussed in Sec. III-A. 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. 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. V-B, 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. 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.

C. Dedicated Depth-Wise 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 [37]. 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. V-B on the *Bottleneck* use-case. Moreover, a pure software execution of such kernels easily becomes a performance bottleneck for computation [8]. To speed up the execution of the depth-wise layers, we therefore

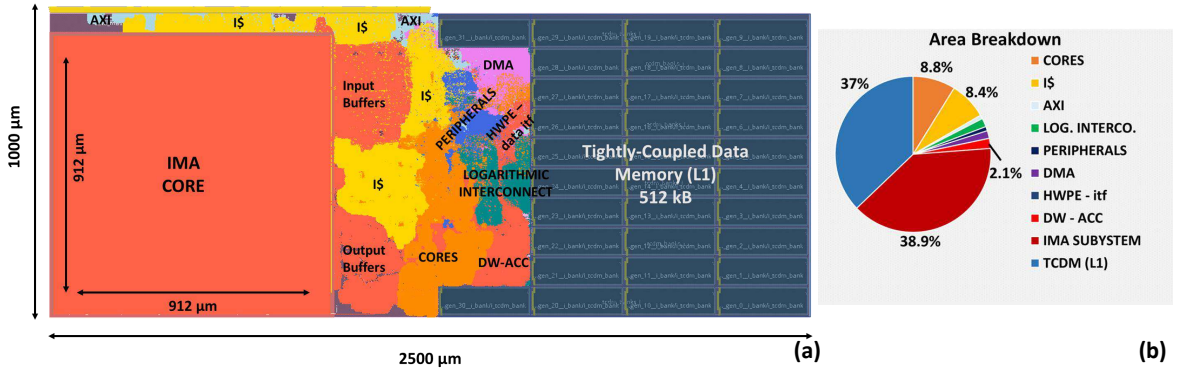


Fig. 6. (a) Placed and Routed design of the heterogeneous cluster. (b) Area breakdown of the system.

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 precision. Fig. 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. 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. 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 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. 5a and Fig. 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.

V. Experimental Results

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

A. Physical Implementation

To characterize the system in terms of area, power, and performance, we implement the cluster using the GlobalFoundries 22nm FDX 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. 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 [27], 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. 6(b). As expected, the IMA sub-system and the 512 kB of TCDM memory

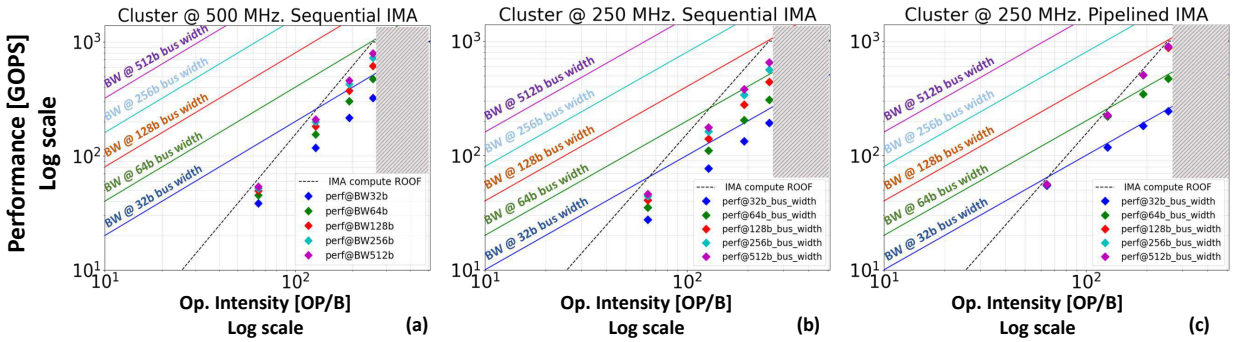


Fig. 7. Roofline plot of the IMA heterogeneous system. The compute roof of the IMA is a diagonal line, quadratically dependent on operation intensity, not on cluster frequency. The intersection of a bandwidth line with the compute roof defines a region where performance points can lay for that configuration. In (a) and (b) cluster runs at 500MHz and 250MHz, with sequential execution of IMA. In (c) it runs at 250 MHz with a pipelined execution model for the IMA.

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 are analyzed with the Synopsys PrimeTime tool and the extracted power is integrated with the power extrapolated for the IMC accelerator [27]. 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.

B. IMC 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 [27]. 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 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.

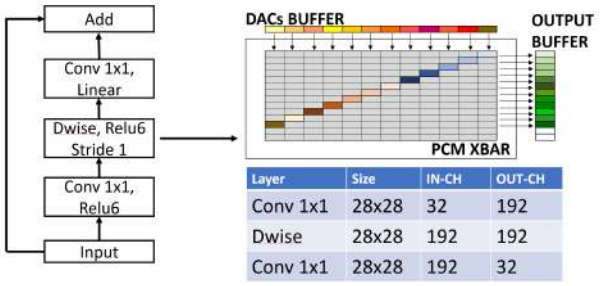
The PULP cluster potentially offers high memory bandwidth towards the TCDM thanks to the tightly-coupled interconnect scheme, at the cost of increased interconnect area (linearly scaling with the bit-width of the system bus), power and timing. To find the width of the system bus able to sustain the IO 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- to 512-bit.

In Fig. 7 we report the outcomes of our exploration as a roof-line plot [38]. 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. 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 Sec. IV-B, 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 used.

Analyzing the scenario where the cluster runs at 250 MHz, Fig.7(b), we observe that higher bus-width (i.e. 128-bit) is necessary to preserve the peak performance of the IMA. 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.7(c) shows that the pipelined solution guar-



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

Fig. 8. Parameters of the *Bottleneck* layer and mapping structure of the depth-wise on the PCM crossbar. Gray rectangles are padding required for computing more than 1 channel per job.

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

C. 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 [36]. The parameters of the selected *Bottleneck* layer are reported in Fig. 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 [33], the in-depth study of which is beyond the scope of this work.

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 [36]. 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 Sec. VI 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. 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 [36]. 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 weights usually account for no more than 10% of the total of a depth-wise based neural network [37] ($\sim 4\%$ 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. 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

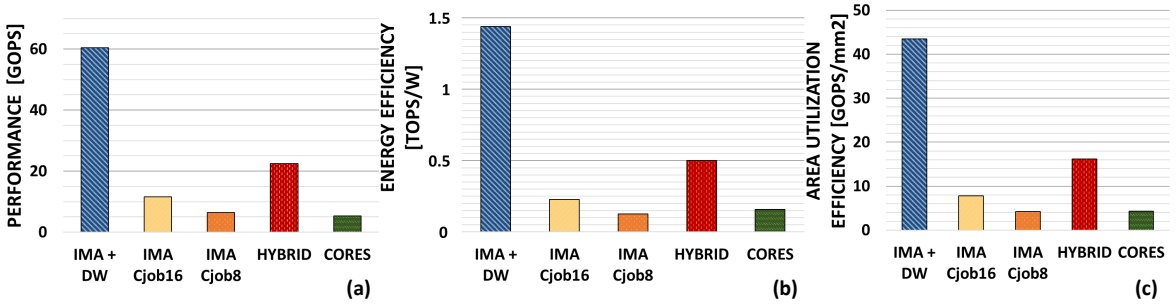


Fig. 9. (a) Performance (in GOPS), (b) Energy Efficiency (in TOPS/W), and (c) Area Utilization Efficiency (in GOPS/mm^2) of the *Bottleneck* layer running on the cluster at 500 MHz with 128-bit wide system-bus. The area efficiency is related to the effective area of the PCM arrays utilized to implement the *Bottleneck* (including padding necessary to map the depth-wise on the IMA).

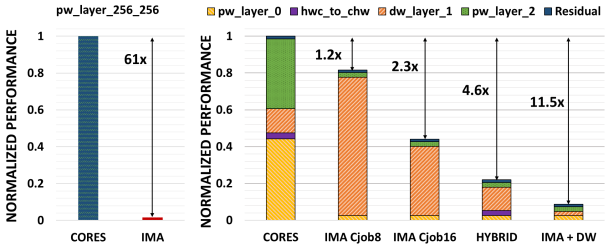


Fig. 10. Normalized Performance, compared to full software implementation (CORES), of point-wise (left) and *Bottleneck* (right) layers. For the right-side analysis, the impact of each layer on the execution of the whole Bottleneck is shown, considering the different computing mapping solutions enabled by the heterogeneous cluster.

model for the IMA; as demonstrated in Secion V-B, 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 surpasses 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 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, achieving 2.7× and 9.2× improvements with respect to 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. 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. 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.

VI. End-to-End MobileNetV2 Inference

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
  $\mathbf{Tiles} \leftarrow [\,]$
  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**
        $\mathbf{Tiles}[n + \text{“\_tile } i\_j\text{”}] \leftarrow (S, S)$
      end for
    end for
    for $j \in [0, n_{\text{tile,w}} - 1]$ **do**
      $\mathbf{Tiles}[n + \text{“\_tile } n_{\text{tile,h}}\_j\text{”}] \leftarrow (h_{\text{rem}}, S)$
    end for
    for $i \in [0, n_{\text{tile,h}} - 1]$ **do**
      $\mathbf{Tiles}[n + \text{“\_tile } i\_n_{\text{tile,w}}\text{”}] \leftarrow (S, w_{\text{rem}})$
    end for
    $\mathbf{Tiles}[n + \text{“\_tile } n_{\text{tile,h}}\_n_{\text{tile,w}}\text{”}] \leftarrow (h_{\text{rem}}, w_{\text{rem}})$
  end for
  for all $n, (h, w) \in$ Tiles **do** ▷ Remove 0-sized tiles
    if $h = 0$ or $w = 0$ **then** remove($\mathbf{Tiles}[n]$)
    end if
  end for;
  $\mathbf{Bins} \leftarrow$ BINBESTFIT($\mathbf{Tiles}$)
  $\mathbf{IMA\_Mapping} \leftarrow$ MAXRECTSBSSF($\mathbf{Bins}$)
  return $\mathbf{IMA\_Mapping}$
end function

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 [37]. 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 work does not support cell re-programming, 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 [27] or row-wise [39] 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 the configuration registers (mapped in different portions of the cluster memory map). Fig. 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 [33] and minimizing the energy one [9] 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. V-C, 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

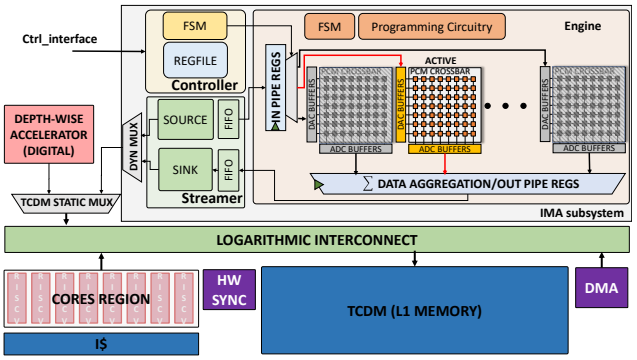


Fig. 11. Overview of the scaled-up heterogeneous architecture. Only one IMC cross-bar can be active at a time.

on the *Maximal Rectangles Best Short Side Fit* bin fitting algorithm[2]. Fig. 12(b) 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* layers of the MobileNetV2 we need 34 IMA cross-bars. As can be seen in Fig. 12(b), 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 $\sim$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. V-C (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. 12(a), whereas in Fig. 12(c) 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 – 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.

[2]We employ the open-source *rectpack* Python library, available at https://github.com/secnot/rectpack [40], to implement the BINBESTFIT and MAXRECTSBSSF functions.

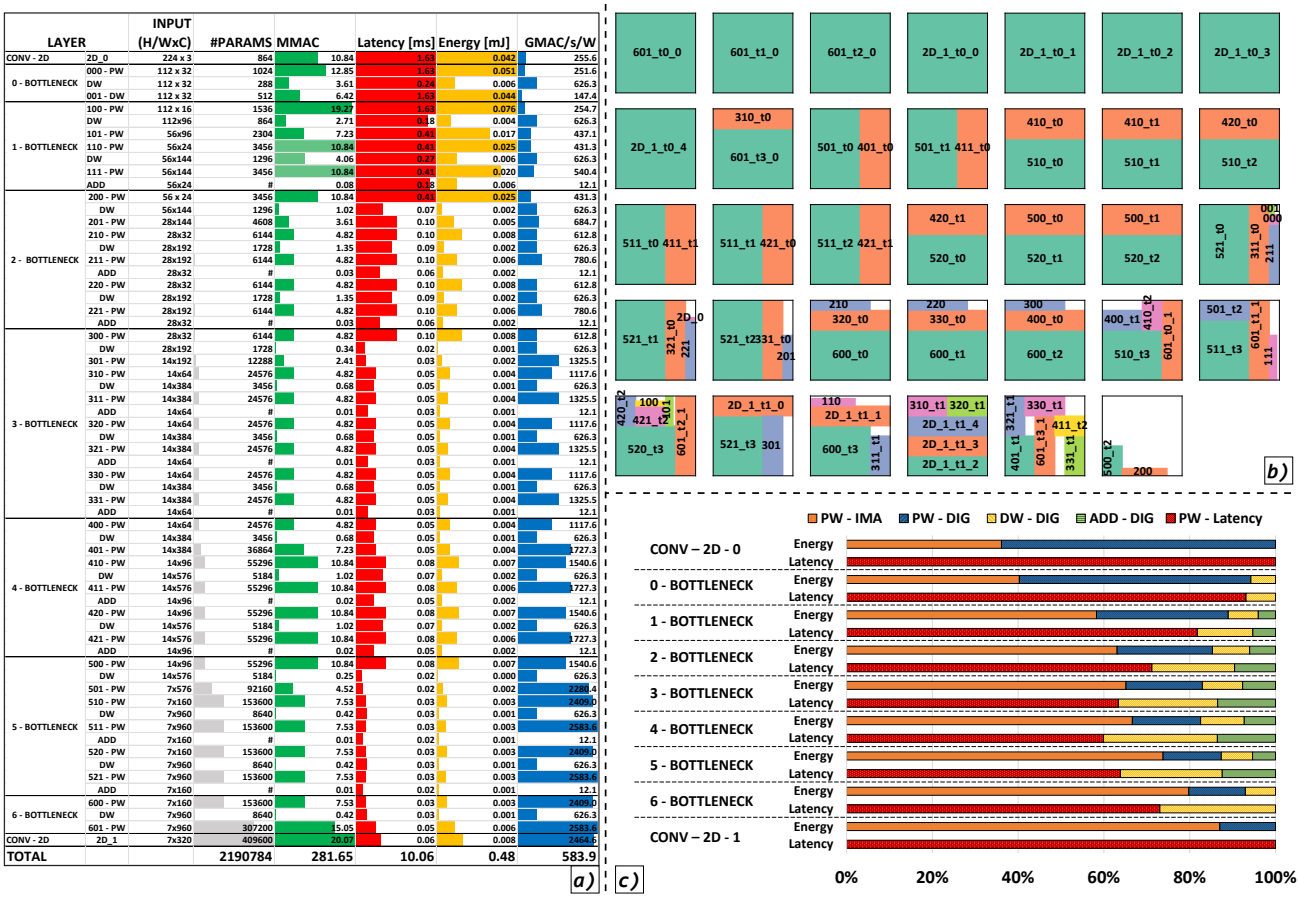


Fig. 12. End-to-end execution of MobileNetV2 on the scaled-up heterogeneous cluster. (a) shows the parameters of each layer, the execution latency (ms), energy (mJ) and efficiency (GMAC/s/W). (b) Tiling algorithm of the layers on the 34 IMAs. (c) Latency and energy breakdown of *Bottleneck* layers.

VII. Comparison with the State-of-the-Art

Tab. I 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 [9], 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 [6], 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 [6] requires extending the architecture with a Flash memory to store the weights of the DNN (with 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 [7] 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 [27], 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. V-C. However, the architecture in [7] 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 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 [7], including a cluster of 4×4 computing in memory units and a network-on-chip for communication which delivers outstanding perfor-

TABLE I
COMPARISON WITH THE STATE-OF-THE-ART.

	[9]	[7]	[31]	[6]	This Work
Tech. node	22nm	14nm	16 nm	65nm	22nm
Area [mm2]	12	3.2	25	13.5	∼30
Cores (ISA)	9x RV32 IMCF Xpulp	None	None	1 RV32 IMC	8x RV32 IMC Xpulp
Analog IMC	None	1x PCM	16x charge	1x charge	34x PCM
Array Rows	None	1024	1152	2304	256
Array Columns	None	512	256	256	256
Digital acc.	HWCE (stand. conv.)	ReLu, activ., im2col	Activ., scaling, pooling	Activ., scaling, pooling	Depth-wise
Peak Perf. [TOPS]	0.032 (ML 8b)	2 (8b-4b)	3 (8b-8b)	0.068 [1] (8b-4b)	0.958 (8b-4b)
Peak Eff. [TOPS/W]	0.61 (8b-8b)	13.5 (8-4b)	30 (8b-8b)	12.5 [1] (8b-4b)	6.39 (8b-4b)
MobileNetV2 inference					
Perf. [inf./s]	10	n/a	n/a	0.23[2]	99
Energy [mJ]	1.19	n/a	n/a	n/a	0.482

[1] Scaled from 1b-1b MVMs performance as explained in [6].
[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 [34]; additional ∼7× improvement factor due to the cluster parallelism [36].

mance and efficiency on MVMs, also the architecture shown in [31] 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 work, we abstract the specific System-on-Chip implementations described in Tab. I to four categories representative of the state-of-the-art, as shown in Fig. 13. We can highlight four different processing models: *i)* analog cores extended with fixed-function digital logic [7], [31] (IMA+ DIG. ACC.), *ii)* analog cores controlled by simple MCU-subsystems [6] (IMA+ MCU), *iii)* IMAs integrated into tightly-coupled clusters of programmable processors [8] (SW+ IMA), and *iv)* the paradigm proposed in this work, 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. 13 shows the results of the exploration, highlighting that for the MobileNetV2 workload, the computational model proposed in this work 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

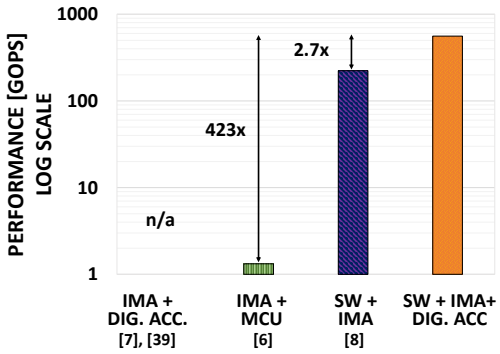


Fig. 13. Performance of the MobileNetV2, on four IMC-based computing models. On the IMA+ASIC it is not possible to deploy the network model, due to architectural limitations.

which they were not intended before fabrication.

We argue that this concept, only demonstrated for MobileNetV2 DNN in Fig. 13 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 [41]. We believe that the approach proposed in this work is a viable way to tackle the performance and flexibility requirements of rapidly evolving modern computer vision pipelines.

VIII. CONCLUSION

Analog in-memory computing (IMC) promises outstanding improvements in energy efficiency on MVM operations. However, to target practical IoT applications IMC arrays must be enclosed in programmable heterogeneous systems, introducing new system-level challenges.

In this work, we explore these challenges 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 FDX technology. Benchmarked on a highly heterogeneous workload such as the *Bottleneck* layer, our solution overcomes software execution of the layer by 11.5× and 9.5× in terms of performance and energy efficiency. We scaled up our system to investigate the challenges and the resources of enabling end-to-end inference of real-world DNNs such as the MobileNetV2, demonstrating 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.

REFERENCES

[1] N. Verma *et al.*, "In-Memory Computing: Advances and Prospects," *IEEE Solid-State Circuits Magazine*, vol. 11, no. 3, pp. 43–55, 2019.
[2] A. Sebastian *et al.*, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.
[3] "Artificial intelligence for a safer, greener and more trusted world," https://www.axelera.ai/, accessed: 2021-12-14.
[4] D. Fick and M. Henry, "Analog Computation in Flash Memory for Datacenter-scale AI Inference in a Small Chip," in *Hot Chips*, 2018.

[5] S. R. Nandakumar *et al.*, "Mixed-Precision Deep Learning Based on Computational Memory," *Frontiers in Neuroscience*, vol. 14, p. 406, 2020. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2020.00406

[6] H. Jia, H. Valavi, Y. Tang, J. Zhang, and N. Verma, "A programmable heterogeneous microprocessor based on bit-scalable in-memory computing," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 9, pp. 2609–2621, 2020.

[7] C. Zhou, F. G. Redondo, J. Büchel, I. Boybat, X. T. Comas, S. R. Nandakumar, S. Das, A. Sebastian, M. L. Gallo, and P. N. Whatmough, "AnalogNets: ML-HW Co-Design of Noise-robust TinyML Models and Always-On Analog Compute-in-Memory Accelerator," 2021.

[8] G. Ottavi, G. Karunaratne, F. Conti, I. Boybat, L. Benini, and D. 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)*. IEEE, 2021, pp. 1–4.

[9] D. Rossi, F. Conti, M. Eggimann, A. Di Mauro, G. Tagliavini, S. Mach, M. Guermandi, A. Pullini, I. Loi, J. 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.

[10] A. Biswas and A. 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*, vol. 54, no. 1, pp. 217–230, 2018.

[11] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature Electronics*, vol. 1, no. 6, pp. 333–343, 2018.

[12] W.-H. Chen, K.-X. Li, W.-Y. Lin, K.-H. Hsu, P.-Y. Li, C.-H. Yang, C.-X. Xue, E.-Y. Yang, Y.-K. Chen, Y.-S. Chang, T.-H. Hsu, Y.-C. King, C.-J. Lin, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, and M.-F. 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)*, 2018, pp. 494–496.

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

[14] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[15] V. Joshi, M. Le Gallo, S. Haefeli, I. Boybat, S. R. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Accurate deep neural network inference using computational phase-change memory," *Nature communications*, vol. 11, no. 1, pp. 1–13, 2020.

[16] Y.-D. Chih *et al.*, "An 89TOPS/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)*, vol. 64. IEEE, 2021, pp. 252–254.

[17] H. Kim, Q. Chen, T. Yoo, T. T.-H. Kim, and B. 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)*. IEEE, 2019, pp. 345–348.

[18] S. Mittal, G. Verma, B. Kaushik, and F. A. Khanday, "A survey of SRAM-based in-memory computing techniques and applications," *Journal of Systems Architecture*, vol. 119, p. 102276, 2021.

[19] S. Yin, Z. Jiang, J.-S. Seo, and M. Seok, "XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 6, pp. 1733–1743, 2020.

[20] J. Yue, X. Feng, Y. He, Y. Huang, Y. Wang, Z. Yuan, M. Zhan, J. Liu, J.-W. Su, Y.-L. Chung *et al.*, "A 2.75-to-75.9 TOPS/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)*, vol. 64. IEEE, 2021, pp. 238–240.

[21] J. Lee, H. Valavi, Y. Tang, and N. 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*. IEEE, 2021, pp. 1–2.

[22] K. Roy, I. Chakraborty, M. Ali, A. Ankit, and A. Agrawal, "In-memory computing in emerging memory technologies for machine learning: an overview," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[23] A. Sebastian, I. Boybat, M. Dazzi, I. Giannopoulos, V. Jonnalagadda, V. Joshi, G. Karunaratne, B. Kersting, R. Khaddam-Aljameh, S. Nandakumar *et al.*, "Computational memory-based inference and training of deep neural networks," in *2019 Symposium on VLSI Technology*. IEEE, 2019, pp. T168–T169.

[24] C.-X. Xue *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)*, 2020, pp. 244–246.

[25] C.-X. Xue, J.-M. Hung, H.-Y. Kao, Y.-H. Huang, S.-P. Huang, F.-C. Chang, P. Chen, T.-W. Liu, C.-J. Jhang, C.-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)*, vol. 64. IEEE, 2021, pp. 245–247.

[26] M. Le Gallo, A. Sebastian, R. Mathis, M. Manica, H. Giefers, T. Tuma, C. Bekas, A. Curioni, and E. Eleftheriou, "Mixed-precision in-memory computing," *Nature Electronics*, vol. 1, no. 4, pp. 246–253, 2018.

[27] R. Khaddam-Aljameh, M. Stanisavljevic, J. F. Mas, G. Karunaratne, M. Braendli, F. Liu, A. Singh, S. 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*. IEEE, 2021, pp. 1–2.

[28] D. Palossi, N. Zimmerman, A. Burrello, F. Conti, H. Müller, L. M. Gambardella, L. Benini, A. Giusti, and J. Guzzi, "Fully Onboard AI-powered Human-Drone Pose Estimation on Ultra-low Power Autonomous Flying Nano-UAVs," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[29] M. Dazzi, A. Sebastian, T. Parnell, P. A. Francese, L. Benini, and E. Eleftheriou, "Efficient pipelined execution of CNNs based on in-memory computing and graph homomorphism verification," *IEEE Transactions on Computers*, vol. 70, no. 6, pp. 922–935, 2021.

[30] P. Houshmand, S. Cosemans, L. Mei, I. Papistas, D. Bhattacharjee, P. Debacker, A. Mallik, D. Verkest, and M. Verhelst, "Opportunities and limitations of emerging analog in-memory compute dnn architectures," in *2020 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2020, pp. 29–1.

[31] H. Jia, M. Ozatay, Y. Tang, H. Valavi, R. Pathak, J. Lee, and N. Verma, "Scalable and Programmable Neural Network Inference Accelerator Based on In-Memory Computing," *IEEE Journal of Solid-State Circuits*, pp. 1–1, 2021.

[32] V. Joshi, M. Le Gallo, S. Haefeli, I. Boybat, S. R. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Accurate deep neural network inference using computational phase-change memory," *Nature communications*, vol. 11, no. 2473, 2020.

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

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

[35] F. Conti and L. Benini, "A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 683–688.

[36] A. Garofalo *et al.*, "PULP-NN: accelerating quantized neural networks on parallel ultra-low-power RISC-V processors," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, p. 20190155, 2020.

[37] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[38] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[39] P. Narayanan *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*. IEEE, 2021, pp. 1–2.

[40] J. 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.

[41] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir, "Inverse kinematics techniques in computer graphics: A survey," in *Computer Graphics Forum*, vol. 37, no. 6. Wiley Online Library, 2018.

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

Gianmarco Ottavi recently started his Ph.D. in electronics Engineerig at the University of Bologna, Italy. After recieving his M.Sc. degree in 2019 he worked as a research Fellow for two years in the department of Electrical, Electronic and Information Engineering (DEI) in Bologna. Its field of research focused on hardware design for efficient inference in low-power systems where he developed specialized ISA extension for RISC-V and system-level implementation of In-memory computing accelerators.

Francesco Conti received the Ph.D. degree in electronic engineering from the University of Bologna, Italy, in 2016. He is currently an Assistant Professor in the DEI Department of the University of Bologna. From 2016 to 2020, he held a research grant in the DEI department of University of Bologna and a position as postdoctoral researcher at the Integrated Systems Laboratory of ETH Zurich in the Digital Systems group. His research focuses on the development of deep learning based intelligence on top of ultra-low power, ultra-energy efficient programmable Systems-on-Chip. His research work has resulted in more than 40 publications in international conferences and journals and has been awarded several times, including the 2020 IEEE TCAS-I Darlington Best Paper Award.

Geethan Karunaratne received his B.Sc. degree in Electronic and Telecommunication Engineering from University of Moratuwa, Sri Lanka in 2014, and the M.Sc. degree in Information Technology and Electrical Engineering from ETH Zurich in 2018. He joined IBM Research – Zurich in 2018, where he is currently a member of In-memory computing group. Geethan is working towards his PhD degree at ETH Zurich. His main research interests are in-memory computing and brain-inspired computing.

Irem Boybat (Member, IEEE) received her BSc degree in Electronics Engineering from Sabanci University, Turkey (2013), and MSc and PhD degrees in Electrical Engineering from Ecole Polytechnique Federale de Lausanne (EPFL), Switzerland (2015 and 2020, respectively). Since 2020, she is a post-doctoral researcher in the In-memory computing group of IBM Research Europe, Switzerland. Her research interests include in-memory computing for AI systems, neuromorphic computing and emerging resistive memory. She was a co-recipient of the 2018 IBM Pat Goldberg Memorial Best Paper Award, 2018 IBM Research Division Award on neuromorphic computing using phase-change memory devices and 2020 EPFL PhD Thesis Distinction in Electrical Engineering.

Luca Benini holds the chair of digital Circuits and systems at ETHZ and is Full Professor at the Università di Bologna. He received a PhD from Stanford University. Dr. Benini's research interests are in energy-efficient parallel computing systems, smart sensing micro-systems and machine learning hardware. He has published more than 1000 peer-reviewed papers and five books. He is a Fellow of the IEEE, of the ACM and a member of the Academia Europaea. He received the IEEE Mac Van Valkenburg award in 2016 and the ACM/IEEE A. Richard Newton Award in 2020.

Davide Rossi received the Ph.D. degree from the University of Bologna, Bologna, Italy, in 2012. He has been a Post-Doctoral Researcher with the Department of Electrical, Electronic and Information Engineering "Guglielmo Marconi," University of Bologna, since 2015, where he is currently an Assistant Professor. His research interests focus on energy-efficient digital architectures. In this field, he has published more than 100 papers in international peer-reviewed conferences and journals. He is recipient of Donald O. Pederson Best Paper Award 2018, 2020 IEEE TCAS Darlington Best Paper Award, 2020 IEEE TVLSI Prize Paper Award.