Horizon 2020 Program (2014-2020)
**FET-Open – Novel ideas for radically new technologies**
**FETOPEN-01-2018-2019-2020**

Architecting More than Moore – Wireless Plasticity for Massive Heterogeneous Computer Architectures [†]

# D4.3: Heterogeneous System-on-Chip

| | |
|---|---|
| Contractual Date of Delivery Actual | 31/03/2023 |
| Date of Delivery | 31/03/2023 |
| Deliverable Dissemination Level | Public |
| Editor | Nazareno Bruschi (UNIBO) |
| Contributors | UNIBO (leader) |
| Quality Assurance | Sergi Abadal (UPC) |

[†]This project is supported by the European Commission under the Horizon 2020 Program with Grant agreement no: 863337.

# Document Revisions & Quality Assurance

| | |
|---|---|
| Deliverable Number | D4.3 |
| Deliverable Responsible | UNIBO |
| Work Package | WP4 |
| Main Editor | Nazareno Bruschi |

## Internal Reviewers

1. Peter Haring Bolívar (UoS)
2. Giovanni Ansaloni (EPFL)

## Revisions

| Version | Date | By | Overview |
|---|---|---|---|
| 1.0.0 | 05/02/2023 | *Nazareno Bruschi (UNIBO)* | First draft |
| 1.0.1 | 06/02/2023 | *Nazareno Bruschi (UNIBO)* | Added introduction |
| 1.0.2 | 06/02/2023 | *Nazareno Bruschi (UNIBO)* | Added SoC description |
| 1.0.3 | 06/02/2023 | *Nazareno Bruschi (UNIBO)* | Added simulation methodology description |
| 1.0.4 | 07/02/2023 | *Nazareno Bruschi (UNIBO)* | Added wireless module description and wireless-based system |
| 1.0.5 | 07/02/2023 | *Davide Nadalini (UNIBO)* | Added training model and ResNet-34 benchmark descriptions |
| 1.0.6 | 07/02/2023 | *Nazareno Bruschi (UNIBO)* | Added inference model and ResNet-18 benchmark descriptions |
| 1.0.7 | 08/02/2023 | *Nazareno Bruschi (UNIBO)* | Added summary |
| 1.0.8 | 08/02/2023 | *Nazareno Bruschi (UNIBO)* | Added inference results |
| 1.1.0 | 13/02/2023 | *Nazareno Bruschi (UNIBO)* | removing placeholders |
| 1.1.1 | 21/02/2023 | *Nazareno Bruschi (UNIBO)* | Added references |
| 1.1.2 | 28/02/2023 | *Nazareno Bruschi (UNIBO)* | Added training results |
| 1.1.3 | 28/02/2023 | *Nazareno Bruschi (UNIBO)* | Finalise for internal review |
| 1.2.0 | 15/03/2023 | *Nazareno Bruschi (UNIBO)* | Integrate reviewer comments |
| 1.2.1 | 15/03/2023 | *Nazareno Bruschi (UNIBO)* | Prepare for new results based on the reviewer's comments |
| 1.2.2 | 20/03/2023 | *Nazareno Bruschi (UNIBO)* | Integrate new architecture figures |
| 1.2.3 | 22/03/2023 | *Nazareno Bruschi (UNIBO)* | Integrate new architecture description |
| 1.2.4 | 22/03/2023 | *Nazareno Bruschi (UNIBO)* | Integrate new wireless description |
| 1.2.5 | 28/03/2023 | *Nazareno Bruschi (UNIBO)* | Integrate new results |
| 1.2.6 | 29/03/2023 | *Nazareno Bruschi (UNIBO)* | Finalise results chapter |
| 1.2.6 | 30/03/2023 | *Nazareno Bruschi (UNIBO)* | Last read |
| 2.0.0 | 31/03/2023 | *Nazareno Bruschi (UNIBO)* | Submitted version |

## Legal Disclaimer

# Executive Summary

The main subject of D4.3 is to describe the massively parallel heterogeneous architecture and the wireless transmission module requirements explored in WiPLASH. This report includes exhaustive results on state-of-the-art tasks in AI, such as inference and training of common Deep Neural Network (DNN)s, in terms of latency, area, and energy consumption compared against a state-of-the-art wired architecture. It first describes the reference architecture illustrating the bottlenecks and proposing the wireless module in different configurations as a solution for the traffic on the wired interconnect, especially going off-chip and to the memories, which limits the overall performance. Then, it describes the simulator used for the entire performance assessment process, the key characteristic and models, and the methodology used to evaluate the wireless results. Subsequently, it outlines the computational models and the benchmarks, highlighting the main advantages and drawbacks of using both technologies to compute the selected applications. Finally, it explores the proposed wireless usage and its evaluation based on the selected benchmarks, representative of an emerging amount of modern DNN workloads. These findings will be exploited to evaluate the feasibility and understand the impact (either positive or negative) of wireless communications at the architectural level, emphasizing domain-specific architectures. This deliverable is related to task T4.1: ”Heterogeneous System On Chip” and T4.4, ”Wireless IO Interface Controller”, both led by UNIBO. All the activities related to these two tasks have been successfully completed.

# Abbreviations and Acronyms

| | |
|---|---|
| **ADC** | Analog-to-Digital |
| **AI** | Artificial Intelligence |
| **CNN** | Convolutional Neural Network |
| **DAC** | Digital-to-Analog |
| **DNN** | Deep Neural Network |
| **DSE** | Design Space Exploration |
| **FPGA** | Field Programmable Gate Array |
| **GEMM** | General Matrix Multiply |
| **HBI** | High-Bandwidth Interconnect |
| **HBM** | High-Bandwidth Memory |
| **HPC** | High-Performance Computing |
| **IFM** | Input Feature Map |
| **IMA** | In-Memory Accelerator |
| **MAC** | Medium Access Control |
| **MIPS** | Millions of Instructions Per Second |

| | |
|---|---|
| **mmWave** | millimeter Wave |
| **MVM** | Matrix-Vector Multiplication |
| **NiP** | Network-in-Package |
| **NoC** | Network-on-Chip |
| **nvAIMC** | non-volatile Analog In-Memory Computing |
| **OFM** | Output Feature Map |
| **PCB** | Printed Circuit Board |
| **PCIe** | PCI Express 4.0 |
| **PCM** | Phase Change Memory |
| **RTL** | Register Transfer Language |
| **SGD** | Stochastic Gradient Descent |
| **SiP** | System-in-Package |
| **SoC** | System-on-Chip |
| **TCDM** | Tightly Coupled Data Memory |
| **TOPS** | Tera Operations Per Second |
| **WP** | Work Package |

## The WiPLASH consortium is composed by

| | | |
|---|---|---|
| UPC | Coordinator | Spain |
| IBM | Beneficiary | Switzerland |
| UNIBO | Beneficiary | Italy |
| EPFL | Beneficiary | Switzerland |
| AMO | Beneficiary | Germany |
| UoS | Beneficiary | Germany |
| RWTH | Beneficiary | Germany |

# Contents

# List of Figures

# List of Tables

# 1. Introduction

With the end of Dennard scaling, multicore processors are now commonly employed, as they present lower power consumption that single processing units. These processors consist of multiple independent cores integrated into a single chip. In theory, these processors have the potential to scale with the number of cores, but, in reality, a high-throughput, low-latency on-chip interconnect is necessary for efficient communication between cores. As the number of cores increases, communication becomes the main bottleneck, limiting multicore chip performance [1].

Recent architectural trends have shifted towards smaller, specialized chips called ”chiplets”, interconnected through a silicon interposer within a System-in-Package (SiP). These chiplets are optimized for specific computations and off-chip transfers dominate the communication requirements at the system level [2] [3] [4] [5]. Within the chiplet, ultra-dense on-chip interconnects are needed to communicate and move towards data-intensive accelerators, crucial for speeding up state-of-the-art Artificial Intelligence (AI) tasks.

In the last decade, the standard interconnect fabric for multicore chips has been Network-on-Chip (NoC), a packet-switched network of integrated routers and wires arranged in a mesh topology. However, this approach has limitations when scaling beyond a certain number of cores, leading to increased latency and energy consumption. The extension of NoC to multi-chip SiPs, known as Network-in-Package (NiP), exacerbates these issues, exploiting multi-hop and high-bandwidth channels [6] [7].

Recent advances in integrated millimeter Wave (mmWave) devices have made wireless networks a promising alternative to traditional wired NoCs/NiPs. The ideal scenario for chip-scale communication would be multiple cores interconnected through a low-latency, multi-Tbit/s, energy-efficient memory. Wireless networks can connect cores to off-chip memories with similar performance to on-chip caches while providing scalable cache coherence.

The performance, efficiency, and flexibility of on-chip and in-package wireless communication present an excellent opportunity to eliminate bottlenecks in current computing systems. The main challenge is identifying where wireless can have the greatest impact and then co-designing the network and architecture to optimally exploit the wireless interconnect while avoiding overloading it. This can be a challenge due to the disruptive potential of wireless innovations in the traditionally incremental architecture domain [8].

For example, in deep learning applications, each layer of a DNN can be mapped to an in-memory computing core that stores the corresponding synaptic weights. However, such accelerators present challenges for communication between cores and interconnection with conventional digital cores. The physical localization of the synaptic weights requires communication over large distances, making wired interconnects prohibitive. DNNs also often require multicasting data from one core to multiple others,

Table 1.1: Collaboration between technical WPs to deliver this report.

| | Description | |
|---|---|---|
| **WP** | **income** | **outcome** |
| WP3 | communication channels specifications such as bandwidth, latency, energy/bit and available frequency and spatial channels | execution traces extracted from simulations |
| WP5 | MAC protocol models and insights | architecture simulator for domain-specific accelerator |
| WP1 and WP2 | area and power for wireless antennas and transceivers | performance targets to get the architectural WP milestones |

making wireless networks a promising solution for eliminating communication bottlenecks in AI applications.

Work Package (WP)4 is naturally connected to the other WPs, and in particular, with WP3 and WP5. Tab. 1.1 summarizes the collaborations on delivering this report. WP3, mainly responsible of the communication channels, provides the characterization of the components of the wireless communication, iterating with WP4 and WP5 to define the specifications to achieve the expected improvements replacing traditional wired channels. While WP4 was providing the specifications, also on behalf of the technology partners (i.e., WP1 and WP2), WP4 has provided the execution traces on the different proposed architectures on different benchmarks to drive the direction on the communication channels research. Together with WP5, WP4 defines the architectures, weekly sharing every step forward on researching the best architectures to propose the disruptive wireless technology replacing part of the wired interconnects and to have a fair comparison with the state-of-the-art. WP5 provided crucial insights on the MAC protocols within the simulator models that WP4 has replicated in this report. Together, we have collaborated on defining the benchmarks, navigating the space of the AI algorithms, and proposing networks and tasks that are mostly used in the specific domain of the goal of WP4.

The remainder of this deliverable is organized as follows. In Chapter 2, we exhaustively detail the proposed reference architecture and the design choices. In Chapter 3, we describe the simulator features and the simulation methodology to model the architecture. In Chapter 4, we describe the execution models and detail the benchmarks used to evaluate the system performance, focusing on mapping and possible bottlenecks for the communications. In Chapter 5, we present the results obtained from our simulator running the abovementioned benchmarks. We detailed the application, system, and communication inefficiency and the design space exploration to find the best trade-off in performance/area/power using graphene-based antennas. Finally, the deliverable is concluded in Chapter 6.

# 2. Massively Parallel Heterogeneous System-in-Package Architecture

Based on exhaustive research at the system level done within the WiPLASH project [9] [10], the proposed architecture of this deliverable is the SiP architecture described in Fig. 2.1. This architecture's high scalability shows the wired infrastructure's inefficiency, especially scaling up the system's complexity from a single node represented here to a multi-node system. Thanks to the modularity of the architecture, we can explore several configurations to reveal the best suited for wireless communications. Moreover, the design choice generates traffic while running the proposed benchmarks that would be affordable only with aggressive optimization efforts on both the software and hardware sides. It also aligns the work with the most recent computer architectures and chip fabrication trends to improve and maintain a reasonable yield. The rest of this chapter details the architecture in both configurations, explaining the benefits of each component.

## 2.1 System-in-Package Architecture

This section presents the proposed heterogeneous many-core SiP architecture. It consists of a chiplet-based architecture, where chiplets are grouped as (i) computing chiplets, composed of multiple heterogeneous (analog/digital) accelerators organized in a hierarchy of quadrants (i.e., C0-C3 and Q0-Q3 in Fig. 2.1), (ii) I/O chiplets to route the chiplet-to-chiplet communications efficiently, and (iii) the memory chiplets, composed of High-Bandwidth Memory (HBM)s (i.e., M0-M3 in Fig. 2.1). This architecture is inspired by state-of-the-art chiplet-based architecture recently developed and launched as commercial products from several market leaders [11] [12]. For wired version, we assume up to 4 computing chiplets and 4 HBMs per I/O hub. With more computing and memory resources needed, the node can be extended via PCI Express 4.0 (PCIe) interface, building a multi-node system. The same has been assumed for the wireless version, but even exploring different design choices, such as with more chiplets connected to the same I/O hub or even without I/O hub, as we will see in Sec. 5.

Every component of the proposed SiP architecture is separately detailed in the rest of this section.

### 2.1.1 Heterogeneous Accelerator

The architecture of the computing chiplet and its components are depicted in Fig. 2.2. It is composed by three main parts: (i) the cluster, (ii) the IMA and the tensor core, and

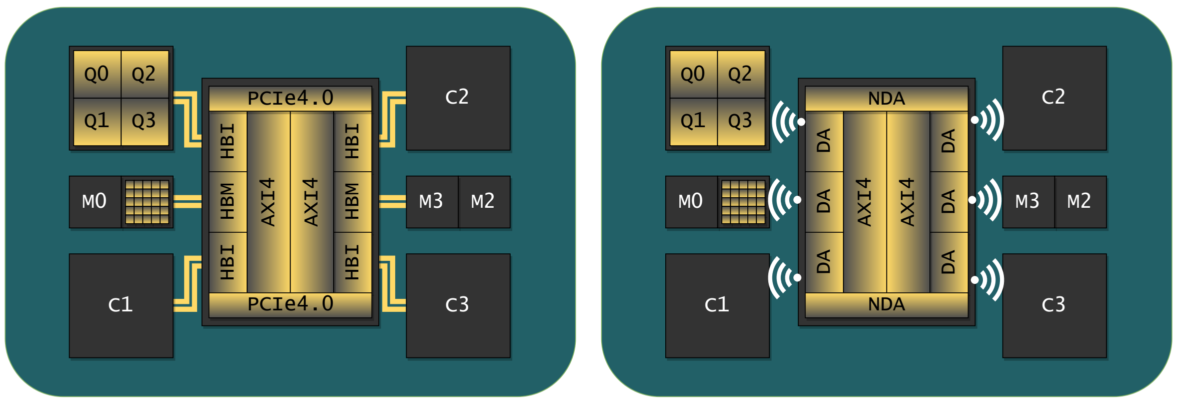


Figure 2.1: System-in-Package architecture. It comprises computing chiplets (i.e., C0-3), memory chiplets (i.e., M0-M3), and an I/O chiplet. On the left, it is the wired version. On the right, it is the wireless one. The substrate is the interposer.

(iii) the hierarchical interconnects. These allow to build the heterogenous multi-cluster computing chiplets.

Tab. 2.1 summarizes the design choices for the entire architecture. It includes the knobs we implemented and evaluated in the Design Space Exploration (DSE) done in Sec. 5.

#### 2.1.1.1 Cluster

The core of the proposed system architecture (i.e., Fig. 2.2B) consists of a heterogeneous analog/digital accelerator called *cluster* (i.e., Fig. 2.2A). Each cluster contains a set of RISC-V cores (CORES) [13], a shared multi-bank scratchpad data memory (L1) enabling Single Program Multiple Data computations, a hardware synchronizer to accelerate common parallel programming primitives such as thread dispatching and barriers, and a DMA for the cluster-to-cluster and cluster-to-HBM communication. Each cluster also includes a non-volatile Analog In-Memory Computing (nvAIMC) IMA (i.e., Fig. 2.2C) sharing the same multi-banked memory as the CORES for efficient communication, similarly to the architecture presented in [14], or a tensor core, called RedMulE, that accelerates compact floating point tensor operations, especially for training and inference of high-precision networks, as described in [15].

#### 2.1.1.2 In-Memory Accelerator

The IMA is built around a Phase Change Memory (PCM) computational memory organized as a 2D array featuring horizontal *word lines* and vertical *bit lines* (Fig. 2.2C). In computational memory, the PCM cells are exploited as programmable resistors placed at the cross points between the word lines and the bit lines. This allows Matrix-Vector Multiplication (MVM) in the analog domain with high parallelism and efficiency. In this work, we assume an MVM to be executed in 130 ns as reported in [16]. At the beginning of each *word lines* and the end of each *bit lines*, Digital-to-Analog (DAC) and Analog-to-Digital (ADC) converters perform the conversion between analog and digital domains, respectively. ADCs and DACs connect to two digital buffers connected to the L1 memory through streamers featuring programmable address generation.

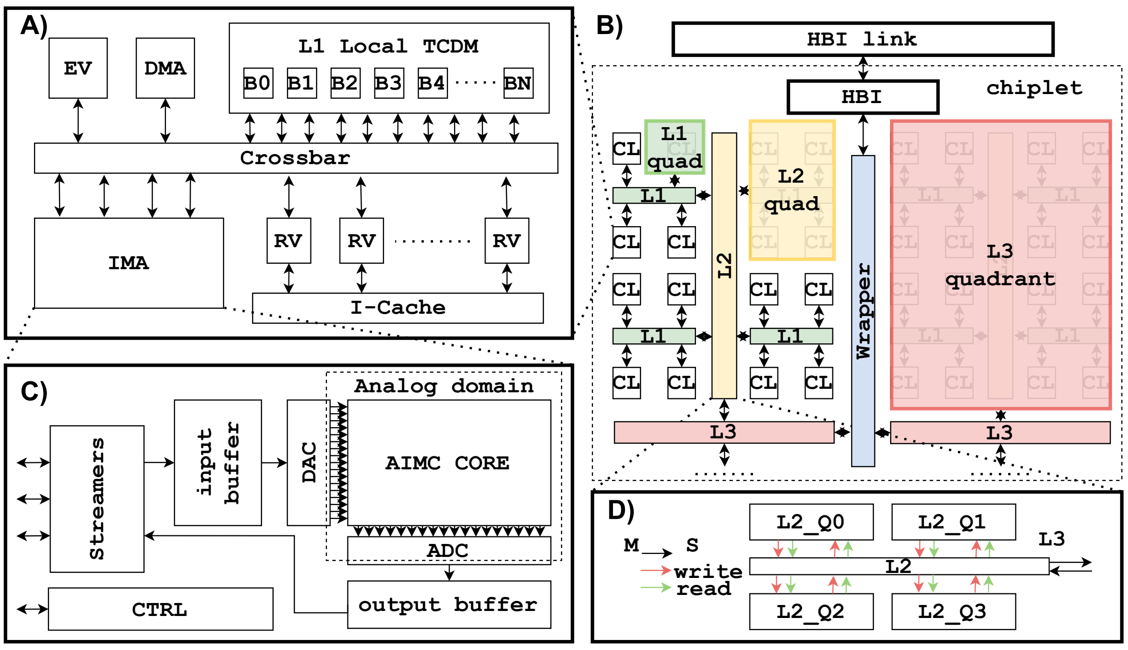


Figure 2.2: A) Cluster architecture. B) Accelerator chiplet architecture. C) IMA subsystem. D) Router model.

### 2.1.1.3 Tensor core

RedMulE [15] is a domain-specific processor designed to accelerate General Matrix Multiply (GEMM) operations. Its architecture is shown in Fig. 2.3a). The core of RedMulE is the Datapath, a 2-Dimensional array of CEs interconnected as shown in Fig. 2.3b). The CEs are organized in $L$ rows, each made of $H$ columns. Within each row, a number of HCEs are cascaded so that each CE computing an intermediate product will pass its result to the next CE. The partial product computed by each row's last CE is fed back as accumulation input of the same row's first CE. The RedMulE Datapath features a design-time configurable number of internal CEs, pipeline registers (P) for each CE, and internal computing precision (FP bitwidth). To feed the Datapath with data, RedMulE includes the Streamer, following the HWPE design strategy. Hybrid FP8 precision formats can be used as an efficient compression scheme to enable DL inference and training on extreme-edge devices.

### 2.1.1.4 On-chip Interconnect

The interconnect infrastructure connecting the clusters consists of a highly parametrizable *hierarchical network* composed of a set of compatible AXI4 *nodes*, as proposed in [17]. The network topology specifies different regions called quadrants connecting groups of clusters: the *Level 1* nodes connect $N_1$ quadrants (clusters), the *Level 2* nodes connect $N_2$ *Level 1* quadrants, and the Level *Level N* nodes connect $N_N$ *Level N-1* quadrants, as shown in Fig. 2.2B for 3 levels. The *Quadrant Factor* for a given *level N* defines the number of quadrants (either clusters or level N-1 quadrants) connected to the AXI node for each level. Clusters feature a master and a slave port, which means that a transaction can either be initiated by the target cluster through its master port or by any other cluster through the target cluster's slave port. The same concept applies to the whole hierarchy of quadrants. In both cases, transactions can

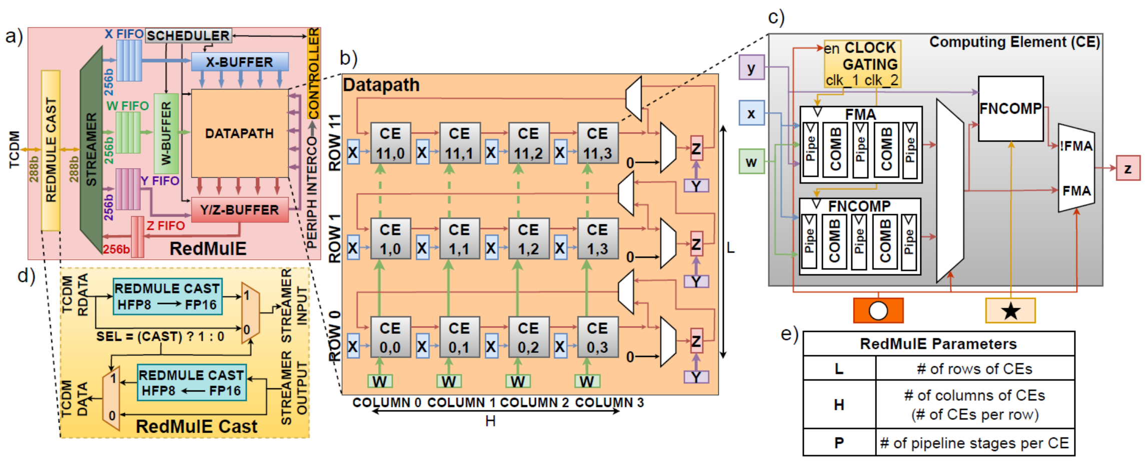
Table 2.1: Architecture parameters

| **System** | |
|---|---|
| Number of nodes | (1, 2, 4) |
| Operating frequency | 1 GHz |
| Number of computing chiplets | (1, 2, 4, 8, 16) |
| Number of memory chiplets | (4, 8, 16) |
| Number of I/O hub chiplets | (1, 2, 4) |
| **Chiplet** | |
| Number of clusters (per computing chiplet) | (512, 256, 128, 64, 32) |
| Total HBM size | 1 GB |
| **Cluster** | |
| Number of IMA | 1 |
| Number of CORES | 16 |
| Number of DMA | 1 |
| L1 memory size | 1 MB |
| **IMA** | |
| Number of streamers ports | 16 (half-duplex) |
| Crossbar size | 256$\times$256 |
| Analog latency (MVM operation) | 130 ns |
| **CORES** | |
| ISA | RISC-V + PULPV2 extensions |
| **DMA** | |
| Local ports | 32 (full-duplex) |
| Max burst size | 2048 bytes |

be either read or write transactions with full support for bursts according to AXI4 specifications (i.e., Fig. 2.2D). The chiplet embeds an interface to be connected to other chiplets via an efficient serial link called High-Bandwidth Interconnect (HBI) [18].

### 2.1.2 I/O Hub

To connect chiplets, the proposed architecture has an I/O hub, a special chiplet that contains all the routing components, and the I/O interfaces, such as the HBM controller and PCIe. The former connects the computing chiplets with the main off-chip memory, and the latter might connect the chiplet nodes together, composing a parametrizable multi-node architecture. Every interconnect technology has been modeled based on state-of-the-art implementations [18] [19] [20] whose main parameters and features are summarized in Tab. 2.2.



| RedMulE Parameters | |
|---|---|
| L | # of rows of CEs |
| H | # of columns of CEs (# of CEs per row) |
| P | # of pipeline stages per CE |

Figure 2.3: a) RedMulE internal architecture; b) RedMulE Datapath microarchitecture; c) RedMulE CE microarchitecture; d) RedMulE Cast module; e) Table with RedMulE design-time available parameters.

## 2.2 Wireless Graphene-based Links

Recent studies suggest that wireless networking can be applied in heterogeneous multi-chip architectures by integrating antennas, transceivers, and computing elements. In this approach, the computing package serves as the wireless propagation medium, bypassing wire routing constraints and offering low latency and scalable broadcast capabilities [21]. Additionally, the use of wideband channels beyond 60 GHz results in the potential for high bandwidths in the tens or hundreds of Gbit/s [22]. This type of interconnect is versatile, as the bandwidth can be shared dynamically among the antennas to meet architecture needs [23]. A defining feature of the wireless interconnect is its seamless support for multicast and broadcast. However, in practice, wireless technologies may experience packet collisions and losses, resulting in decreased effective bandwidth due to re-transmissions [21], and the MAC protocol assumes a crucial role to guarantee high performance [**?**].

In this work, we extensively use wireless properties and especially the availability of decoupled frequency and spatial channels. We assume simple, low-order modulations with spectral efficiency of 1 b/s/Hz. This means that for each Hz of spectral bandwidth, we obtain 1 bit/second in data rate. In this scenario, the number of frequency channels will be calculated by taking the entire spectrum considered and dividing it into chunks of, say, 20 GHz (20 Gb/s). However, this only concerns the creation of multiple links operating in parallel and does not imply an increase in the overall bandwidth. This means that if the total available bandwidth is 200 GHz (equivalent to a rate of 200 Gb/s), the fact that I divide it into several frequency channels does not increase the overall system bandwidth, which stays at 200 Gb/s. On the other hand, the number of spatial channels is estimated based on the capability of creating antenna arrays that can direct/focus energy in specific parts of the chip. This assumption allows multiplying the entire system bandwidth by the number of spatial channels. So, if my available bandwidth is 200 GHz (equivalent to a rate of 200 Gb/s) and I have 5 spatial channels, my overall system bandwidth is 1000 Gb/s.

This work proposes a wireless transmission module composed of two-frequency channel antennas for simultaneous transmission and reception, maximizing traffic management. The module contains all the components needed to interface digital and analog parts, supporting outstanding transactions. This keeps the asynchronicity feature allowing non-blocking transfers. Antennas in the system share the same propagation medium and MAC protocols implement the mechanisms required for synchronization, safe access, and collision handling in a shared transmission medium, as is the case of wireless transmission links. Here, we have implemented the exponential backoff protocol, emulated in the MAC layer of the wireless transmission module, a random access protocol that manages collisions by retransmitting lost data. When employing it, a transceiver receiving a transmission request waits randomly within a dynamically-sized window. The window size is dynamically adjusted according to the network load to balance the transmission rate and the probability of causing a collision. When a collision is detected by the physical layer, the window size increases, and the transceivers involved reschedule the corresponding transmissions. Similarly, when a successful transmission is observed, the length of the window is reduced. The implemented exponential backoff protocol allows for the growth rate modification, window size reduction, and tuning of the maximum window size.

The proposed architecture replaces the wired interconnect infrastructure at different levels to explore every design space. Considering the component size, wireless antennas have been posed in the following conditions: replacing (i) only the PCIe communications, (ii) the chiplet-to-chiplet communications on the same node and between different nodes, to connect (iii) all the chiplets to the same I/O hub, building a single-node architecture, and to connect (iv) all the chiplets together, without the I/O hub to centralize the communications, as summarized in Tab. 2.3. In this table, with single-single we refer to a situation where we have a single frequency channel in a single spatial channel (i.e., they have the full bandwidth but can conflict). With multi-single we refer to a situation with multiple frequency channels in the same spatial channels (i.e., they share the bandwidth but without conflicting). Ultimately, with multi-multi channels we refer to a situation of multi frequency channels in multi spatial channels (i.e., sharing the bandwidth as group).

These solutions aim to exhaustively explore the design space, enabling the usage of architectures that are not feasible at all with the traditional wired technology without several area and power increments. At system level, we have exploited different spatial channels (i.e., up to three) to decouple chiplet to chiplet, chiplet to HBM, and chiplet to node communications, while we have exploited different frequency channels to reduce the contentions between chiplets communications. These assumptions require careful node design rules to distance the components to avoid interference between the abovementioned communications.

## 2.3 Chiplet integration

Fig. 2.4 describes the chiplets integration and the off-chip interconnects. The topmost figure, in particular, refers to a traditional wired integration, where every SiP is packaged and posed on a PCB substate. The silicon interposer in every package routes the different chiplet's communications. The off-chip communications are routed via the PCB. in these architectures, the chiplets are usually implemented in an advanced

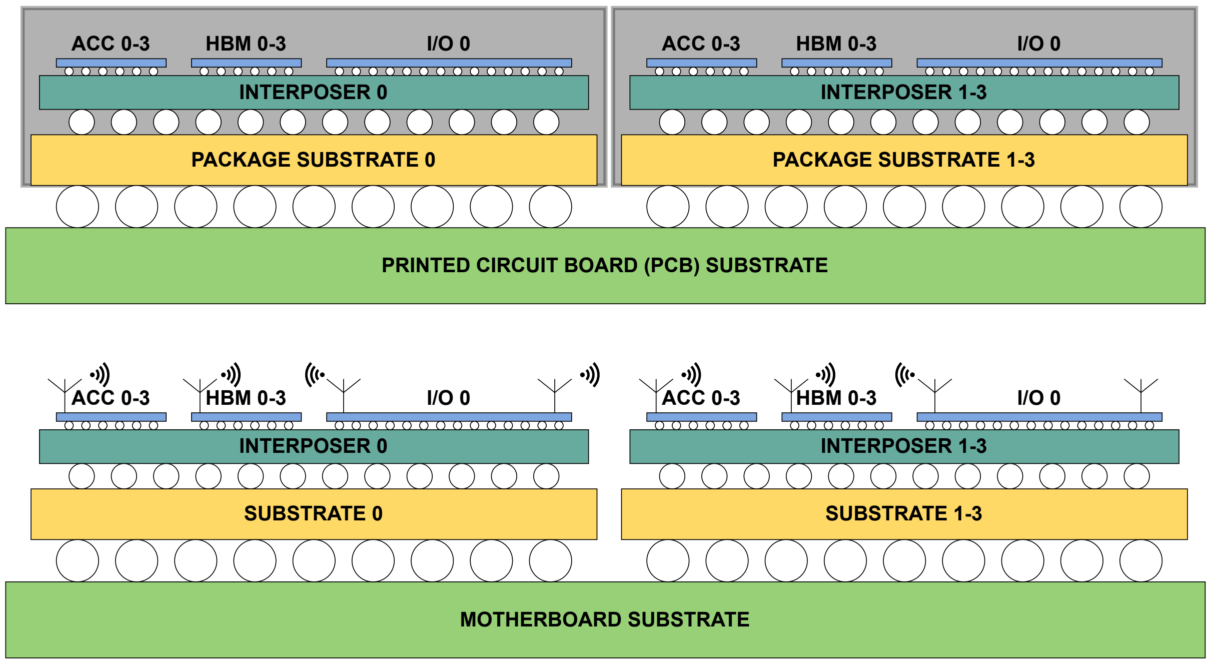


Figure 2.4: Chiplet integration in both cases. In the wired version (i.e., top), every interposer is packed and posed on the PCB. In the wireless version (i.e., bottom), the interposed are non-packed to leverage the wireless long-range communications. The substrates are then connected to motherboard sockets.

technology node. In contrast, the interposer that contains the chip-to-chip links is implemented in less scaled technology. This helps to contain the fabrication costs and the process yield. The bottom part of Fig. 2.4 refers to the version of the architecture using the wireless channels as both on- and off-chip interconnect. In particular, the structure would be the same for the SiP without the package. In fact, to allow wireless communication between nodes, the radiation should have a proper propagation medium, and the SiP package would be isolation. Therefore, in the case of wireless interconnect the SiPs are non-packaged and can communicate with each other in the communication range. Important to notice is that the communication between nodes is allowed only between I/O hub chiplets, de-facto concentrating the communications on a specialized component.

Table 2.2: Interconnect parameters

| | | | |
|---|---|---|---|
| **On-chip Interconnect** | | | |
| cluster-to-cluster | AXI 4 full-crossbar | | |
| I/O | AXI 4 full-crossbar | | |
| **AXI 4 full-crossbar** | | | |
| Direction | full-duplex | | |
| Bandwidth | 512 Gbps | | |
| Latency | 4 cycles | | |
| Lane | 1 | | |
| **Off-chip Interconnect** | | | |
| chip-to-chip | HBI | | |
| chip-to-mem | HBM | | |
| node-to-node | PCIe 4.0 | | |
| | **HBI** | **HBM** | **PCIe 4.0** |
| Direction | full-duplex | full-duplex | full-duplex |
| Bandwidth | 112 Gbps | 512 Gbps | 32 Gbps |
| Latency | 100 cycles | 100 cycles | 500 cycles |
| Lane | 2 | 2 | 32 |
| **Integrated Wireless Communications** | | | |
| Maximum distance | 30 cm | | |
| Bandwidth | (64, 128, 256, 512) Gbps | | |
| Latency | 1 ns | | |
| Frequency channels | up to 16 | | |
| Spatial channels | up to 4 | | |
| **Wireless Transmission Module** | | | |
| Number of antennas | 2 | | |
| Number of frequency channels | 2 | | |
| **MAC Layer** | | | |
| Protocol | exponential backoff | | |
| Overhead | 1 cycle | | |
| **Exponential backoff** | | | |
| Contention window size | 256 cycles | | |
| Max backoff exponent | 16 | | |

Table 2.3: Architecture configuration.

| | Communication | | | | |
|---|---|---|---|---|---|
| **Technology** | **On-chip** | **Wireless line** | **In-package** | | **Off-package** |
| wired | AXI4 | none | HBM | HBI | PCIe |
| wireless 0 | AXI4 | data, ctrl, insn | HBM | HBI | wireless single-single channels |
| wireless 1 | AXI4 | data, ctrl, insn | HBM | wireless multi-single channels | wireless single-single channels |
| wireless 2 | AXI4 | data, ctrl, insn | wireless multi-single channels | wireless multi-single channels | wireless single-single channels |
| hybrid 0 | AXI4 | data | wireless multi-single channels | wireless multi-single channels | wireless single-single channels |
| wireless 3 | AXI4 | data, ctrl, insn | HBM | wireless multi-multi channels | none |
| wireless 4 | AXI4 | data, ctrl, insn | wireless multi-multi channels | wireless multi-multi channels | none |
| hybrid 1 | AXI4 | data | wireless multi-multi channels | wireless multi-multi channels | none |
| wireless 5 | AXI4 | data, ctrl, insn | wireless single-single channels | wireless single-single channels | none |

# 3. Simulation Infrastructure and Methodology

We modeled the proposed architecture by extending an open-source simulator named GVSoC [24] meant to simulate RISC-V-based clustered multi-core architectures. It is a C++ event-based simulator featuring a simulation speed of 25 Millions of Instructions Per Second (MIPS) and an accuracy of more than 90% compared to a cycle-accurate equivalent architecture when simulating a full DNN in a single cluster, as reported in [24].

## 3.1 Time Modeling

In addition to the functional top accuracy, GVSoC has timing models for every relevant activity, such as instructions execution, DMA transfers, and memory accesses. It can emulate the actual system execution and provides a comprehensive set of statistics using hardware features or dedicated profiling tools. In this context, a global *time engine* manages the overall time (at the picosecond scale). A *clock engine* models a clock source as a forward monotone counter associated with a queue of related clock events, which are generic workloads associated with a specific clock cycle. Each event includes a data payload and a pointer to an associated callback function. The clock engine defines a time window ($T_w$) in which the close enough events are included in a circular buffer. The execution of these events is done cycle-by-cycle, and simultaneous ones are executed sequentially using a non-ordered queue. The circular buffer can be fed with new events at any time if its execution cycle is inside $T_w$. If it is greater than $T_w$, the clock engine stores the event information in an ordered queue that is read every time a circular lap is completed. A different frequency can be set for each clock engine, enabling the integration with the global time engine. The mapping of clock events into the global time domain is performed by multiplying the clock period by the difference between the current clock counter and the clock time associated with the event.

## 3.2 Performance Assessment

To extract the execution and timing information, every core models a set of performance counters as the real hardware, measuring the events. These counters could be activated simultaneously, having one counter for each metric and saving much simulation time. Timing information is also stored in a complete set of system traces. Every trace could be either an event or if the event is quite critical, could there be more than one trace per event. The same system traces that could be used to extract timing information are beneficial to extract debugging information such as the content of the registers during the execution.

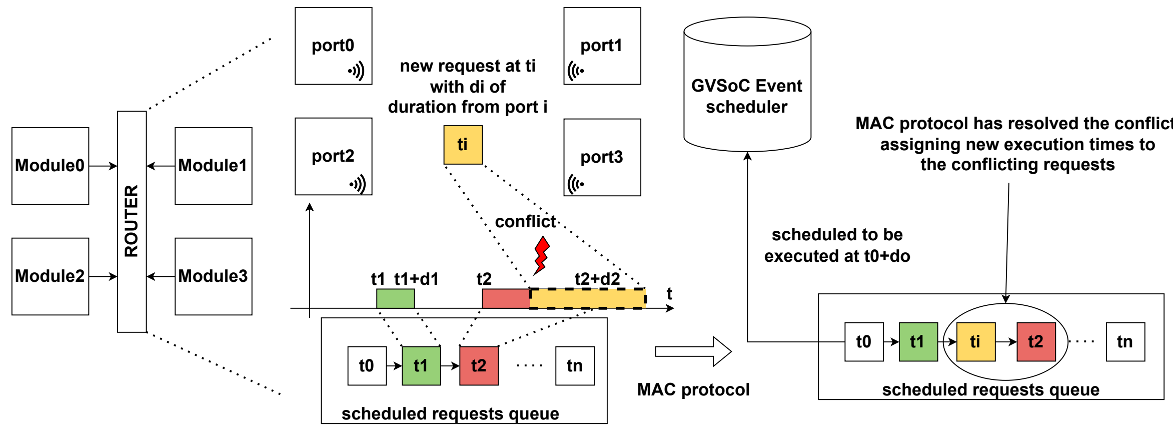

Module0
Module1
Module2
Module3
ROUTER
port0
port1
port2
port3
new request at ti with di of duration from port i
ti
conflict
t1 t1+d1
t2
t2+d2
t
t0
t1
t2
tn
scheduled requests queue
MAC protocol
GVSoC Event scheduler
scheduled to be executed at t0+do
MAC protocol has resolved the conflict, assigning new execution times to the conflicting requests
t0
t1
ti
t2
tn
scheduled requests queue

Figure 3.1: Wireless medium model in GVSoC. Wireless medium schedules events in the GVSoC event engine when a new communication is established. Contentions are detected and solved by the MAC protocol implemented in the model.

## 3.3 Methodology

The main components integrated into the simulator are the IMA and the interconnect infrastructure extending the simulator's capabilities towards many-core accelerators (i.e., up to 512 clusters and 8192 RISC-V cores). The IMA is integrated into the cluster as a master of the cluster crossbar. All the components of the IMA have been modeled, including the input and output buffers and the streamers. At the system level, the interconnect infrastructure has been modeled as a set of parametric router components with configurable data width, latency, and the number of master and slave ports combined together to create the topology described in Fig. 2.2D. The interconnect component in GVSoC has been modeled as a router, which models the routing of the generic transactions. The component models the ports and has an associated latency for routing from one component to another. It models the contentions depending on the type of technology, topology and features, adding a delay to the forwarded transaction. Every interconnect component is configurable, specifying, for example, the number of ports, the latency, and the mapping. Tab. 2.1 and 2.2 describe the configuration parameters of the platform used in this work. All the modules in the simulator have been calibrated using the cycle-accurate Register Transfer Language (RTL) and Field Programmable Gate Array (FPGA) equivalent, excluding the wireless transmission module. 256$\times$256 IMA size has been used since it has been demonstrated in more works and shows better technological feasibility at this time [16].

Wireless transmissions have been modeled as described in Fig. 3.1. We have focused this work on the random access MAC protocol described in Sec. 2.1. The wireless medium is modeled using a particular router, where the modules connected share the same spatial and frequency channels. Every port can produce a request for any other port independently and simultaneously if they do not start and finish at the same ports (i.e., from port0 to port1 and from port2 to port3). In fact, the transmission from an antenna automatically blocks the antenna at the receiver side. Every request has a timestamp to be scheduled and a duration accordingly to the bandwidth and the packet size. The model stores the requests in a scheduled requests queue whether they are not conflicting in time. The scheduled queue is kept ordered to maintain coherency in time. The first scheduled request is always scheduled in the GVSoC event engine at the correct time (i.e., $time + duration$). Whether a conflict occurs, like for the case depicted in Fig. 3.1, the model calls the MAC protocol to resolve the conflict and reschedule the transmission. The conflicts handler manipulates the timestamps of the conflicting requests accordingly to the particular protocol until all the conflicts (also the ones recursively generated) have been resolved and every request has been correctly rescheduled.

# 4. Computational models

In this chapter, we detail the implemented computational models to execute end-to-end Convolutional Neural Network (CNN) inference and training on the proposed device, describing the main characteristics of the execution and data flow, the synchronization between the involved engines, and the computation paradigms.

## 4.1 Inference

This section presents the computational model of the proposed massively parallel heterogeneous architecture computing an end-to-end inference, detailing its main characteristics: Layer Mapping, IMA execution, Pipelining, Data Tiling, and Self-Timed Execution Flow.

### 4.1.1 Static Layer Mapping

According to the computational model of the proposed architecture, each layer of a DNN is statically mapped to a certain number of clusters, while the Input Feature Map (IFM)/Output Feature Map (OFM) are streamed from producer to consumer clusters. Fig. 4.2B shows the mapping of the ResNet-18 on the architecture, where each node of the graph in Fig. 4.2A represents a CNN layer, grouped by color according to the IFM dimensions, and every layer is mapped on different clusters of the system, as shown in Fig. 4.2B. The number of clusters used to map a specific layer depends on the number of parameters of the layer. For example, *Layer 22* features 2.3M parameters, requiring 40 clusters for the mapping, assuming each 256x256 IMA can store 64K parameters.

### 4.1.2 IMA Execution

As described in Sec. 2.1.1, the IMA subsystem communicates directly to the L1 of the cluster, acting as a master of the TCDM interconnect. Assuming DNN parameters of a specific layer are being pre-loaded to the non-volatile array, IMA execution is composed of three distinct phases, as shown in Fig. 4.3. *Stream-in* fetches the IFM of the layer and moves them to the input buffer of the IMA. *Compute* performs the input data conversion by the DACs, the analog MVM execution on the crossbar, and the ADCs conversion. *Stream-out* moves the output digital MVM result from the output buffers to the L1 memory. Input and output buffers are duplicated to enable double buffering, completely overlapping the cost of transfers between the L1 and the buffers with the computation, maximizing the computational efficiency of the accelerator.

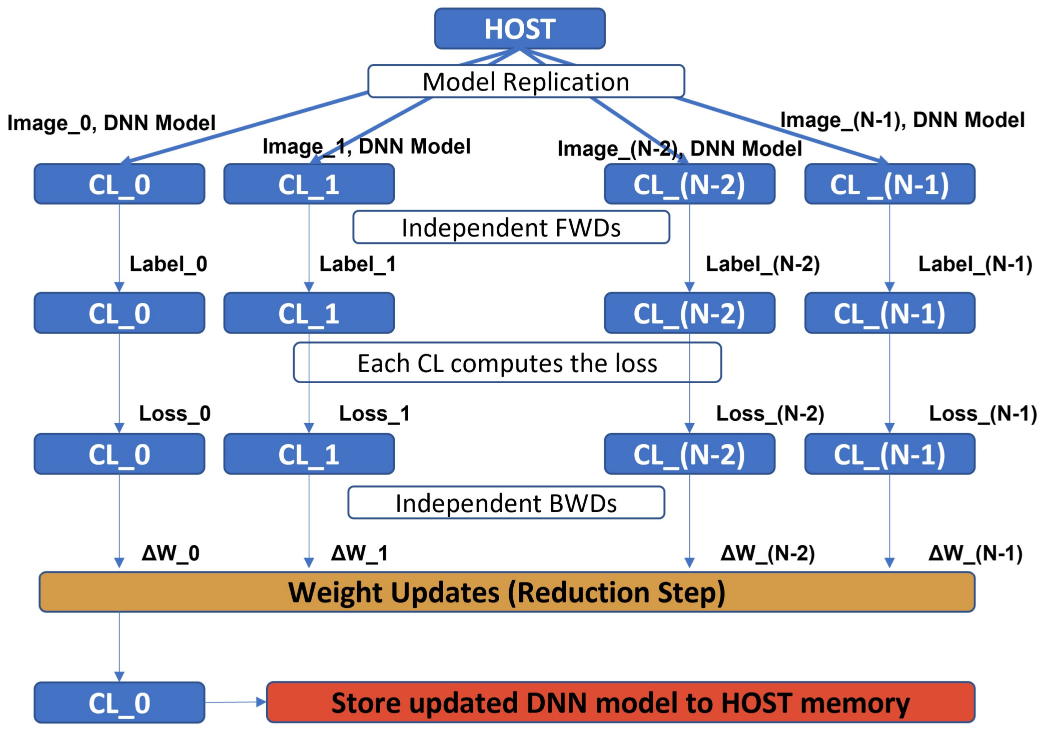


Figure 4.1: Distributed data-parallel approach used for the ResNet-34 training on the proposed massively parallel system. Batches are executed independently until the weights update step, where they are collected and averaged before storing the final trained model. The entire execution is repeated till the end of batches.

### 4.1.3 Pipelining

When the inference starts, the IFM of the first layer is streamed into the first set of clusters which process it generating the OFM, which is then passed to the second set of clusters and so on. Assuming the possibility of having large batches of images allows for the creation of the software pipeline described in Fig. 4.2C, where different chunks of data are processed by a different set of clusters simultaneously, fully overlapping the data movements (i.e., in charge of the DMA) with the computation (i.e., in charge of IMA and/or CORES). Ensuring that all pipeline stages execute in the same amount of time is essential when creating such a pipeline structure.

### 4.1.4 Data Tiling

To fit IFM/OFM of large DNN models within the limited memory resources of the clusters (1 MB of L1 memory is assumed in this work), we split IFM/OFM into smaller chunks of data called *tiles*, processed by the clusters as soon as the input data is transferred to the L1 memory. In particular, data tiling is always performed along the $W_{in}$ and $W_{out}$ dimensions for input and output, respectively. In this work, we assume a static tiling strategy, and $W_{in/out}$ implicitly defines the batching dimension. Therefore, the batches are composed of vertical slices of IFM/OFM. The other dimensions ($C_{in}$

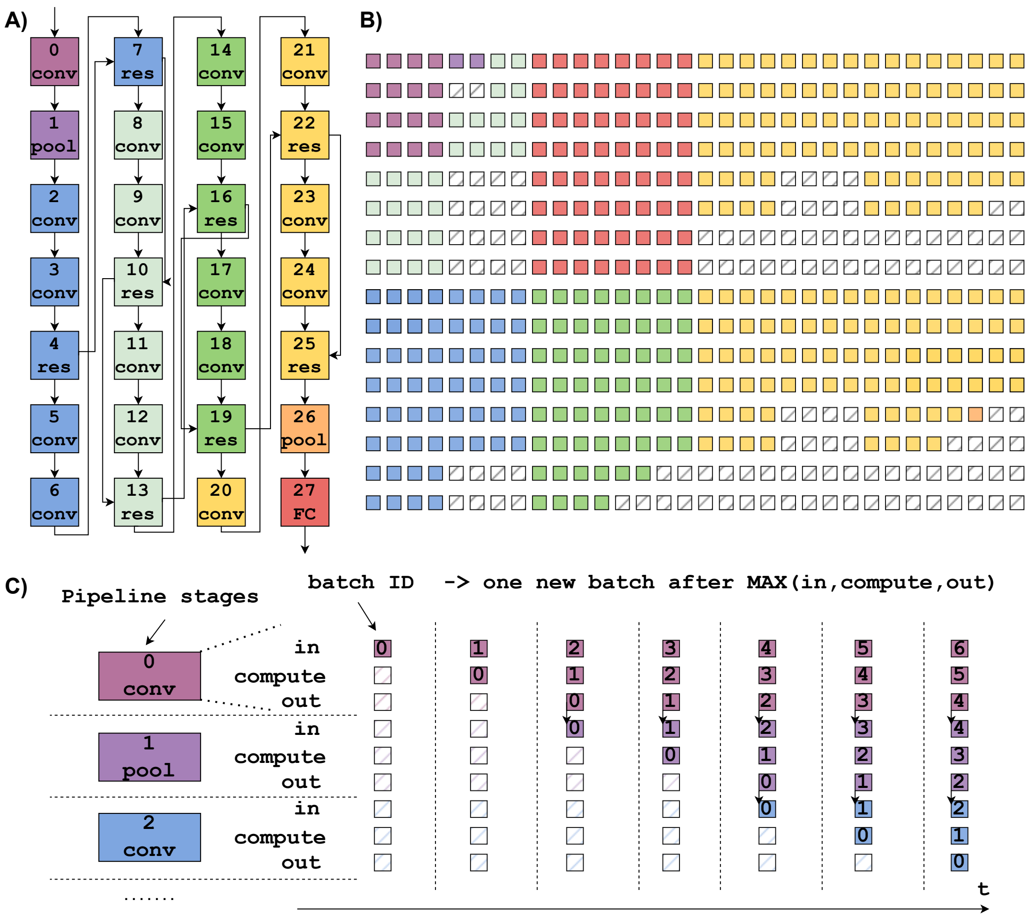


Figure 4.2: A) Directed Acyclic Graph (DAG) of the ResNet-18 execution. B) Mapping example on 512 clusters. C) High-level description of pipelining computational model.

and $H_{in}$) are, when necessary, tiled in other clusters to fit the memory requirements (parameters mapping) or to speed up the computation (*parallelization*).

### 4.1.5 Self-Timed Execution

To implement the pipeline between the tiled structure described in Sec. 4.1.4, we exploit a data-flow self-timed execution model. Computation in a cluster can be performed by the CORES, IMA, or both in parallel. While software execution on the CORES is synchronous, IMA execution is managed asynchronously (like DMA transfers). A cluster can perform a certain computation whenever three conditions are satisfied: a) Chunk N+1 from the *producers* can be loaded to the L1 memory, b) the *consumers* are ready to accept the output data of chunk N-1, c) both IMA and CORES are free to compute chunk N. If all the conditions are satisfied, the new iteration can start with the following execution flow: 1) the CORE0 (i.e., master core) first waits for the events from the input and output DMA channels and IMA, 2) the CORE0 configures and triggers I/O DMA channels and IMA for computation of next tile 3) digital processing is performed in parallel on the CORES. 4) All the CORES go to sleep, waiting for the events described in point 1).

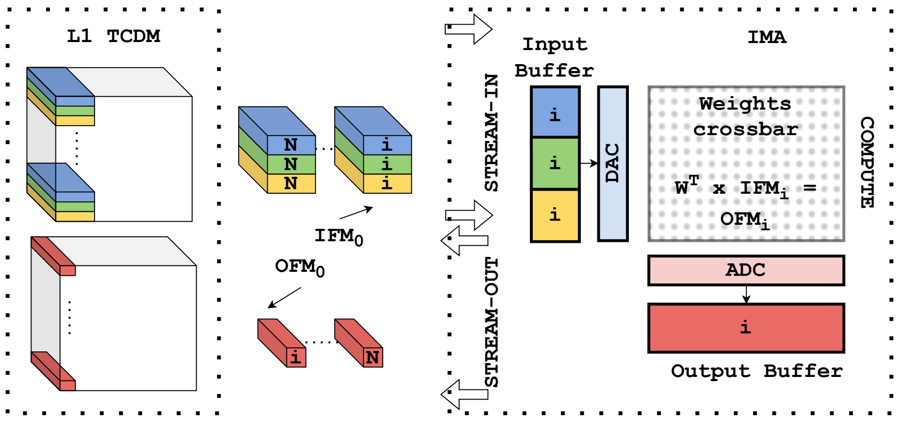


Figure 4.3: IMA execution model. Input and output data are stored in the TCDM. Parameters have been offline pre-stored in the IMA crossbar. The datapath autonomously feeds and drains data from/to outside after a lightweight streamers programming phase by one RISC-V core.

## 4.2 Training

Adopting a distributed data-parallel approach, we distribute the same weight parameters (i.e., the global model) on each cluster, while the input image for each cluster is different. Each cluster independently receives and processes a single image sequentially, performing layer by layer both the forward pass (common inference) and the backward pass to calculate the gradient of the loss function with respect to its input image. Then, the gradients of each layer weight tensor of each layer are collected and averaged to compute the global update performed on each layer of the model. Finally, the global model is updated and stored in memory for the next iteration. In our approach, Cluster 0 orchestrates the execution of the training process. Fig. 4.1 depicts the distributed data-parallel computational model described above, while the following sections detail every component singularly.

For this application, the cluster's engines involved in the execution are DMA, CORES and tensor core since the nvAIMC performs well only during the inference, where the weights have been trained.

### 4.2.1 Forward

During the forward step, the input image for each cluster is processed layer-by-layer. Considering a single cluster, the input image is loaded in the L1 memory by means of the Cluster DMA. Then, the first layer weights are loaded using the same DMA. During the DMA calls, which are put in the DMA queue, the cluster waits for the completion. Then, the forward primitive of the current layer is called and is performed on the loaded data by parallelizing its computational workload on the CORES or tensor core. Then, the result is stored back into the global memory for the backward pass, using the DMA, and the next layer is processed by loading the new weights. The forward pass ends with the last layer, which stores the last layer's prediction in the global memory. The CORES or tensor core executes in parallel on different data and independently. At the end of its execution, each cluster reaches a software barrier to synchronize with the others before performing the next step.

### 4.2.2 Backward

Before the backward step, the loss function ($L$) with respect to the ground truth label is computed by each cluster's CORE 0. In this step, each cluster independently loads the label from the global memory, then computes the network's loss and output gradient ($\frac{\partial L}{\partial o}$). Then, a synchronization barrier is launched. After the barrier, a backward step is performed to compute the weight gradient ($\frac{\partial L}{\partial w}$) and input gradient ($\frac{\partial L}{\partial i}$) for each layer. Starting from the last layer and going to the first, each cluster independently starts by loading its output gradient and input through its DMA. This is done to compute the gradient of the weights. Then, the gradient is stored using the DMA, and the weight tensor of the current layer is loaded to compute the input gradient of the current layer. Even in this case, both the weight and the input gradient computational primitives are parallelized on the CORES, and they adopt a naïve approach to minimize the L1 memory occupation. After the gradients are computed for each layer and the first layer is reached, the backward step is complete, and a cluster synchronization barrier is launched to synchronize all clusters before performing the weight update (again, the flow for each gradient is: load data with DMA, wait, compute primitive, store result with DMA, wait). Note that no input gradient is calculated when the first layer is reached, as they are required to backpropagate the error to the previous layer.

### 4.2.3 Update

During the backward step, all the gradients of the weights are computed to optimize the model, updating each layer's parameter using the pre-computed gradient with a given learning rule. In the case of Stochastic Gradient Descent (SGD), $w = w – h * dw$, where w are the weights, h is the learning rate (hyperparameter), and $dw$ is the weight gradient for the given layer. The weight update process is performed by first averaging the gradients of the whole batch – i.e., all the gradients computed by each cluster are collected, summed, and divided by the number of clusters. The update is performed as a binary tree to speed up the process.

In our approach, Cluster 0 is the master. We consider a set of $N_c$ clusters. The procedure is performed layer-by-layer, and in the beginning, each Cluster loads its weight gradient tensor computed during the last backward step. During each iteration $T = 1, .., log2(N_c)$, each Cluster with index $I_c$ multiple of ($\frac{N_c}{(T-1)}$), including Cluster 0, collects the gradient of the following Cluster – i.e., the one with index $I_c + T$ – and accumulates it on its gradient value. The other clusters wait for the completion of the task. During the first iteration, all clusters load the weights. During the second, Cluster 0 collects the weight gradient of Cluster 1, Cluster 2 collects the weights of Cluster 3, and so on. Then, Cluster 0 sums the gradients it just collected on its local accumulator, and Cluster 2, 4, and so on do the same. During the fourth, Cluster 0 collects the weight gradient from Cluster 2, Cluster 4 from Cluster 6, and so on. The process continues until only Cluster 0, and Cluster Nc-1 are active, and Cluster 0 collects Cluster Nc-1's accumulated weight gradient (iteration $Tf = log2(N_c)$). Finally, Cluster 0 owns the accumulation of all weight gradients from all Clusters on a single accumulator. Therefore, it divides each weight gradient element by $N_c$ to complete the average.

During each iteration, each Cluster loads the data from the other Cluster using its DMA, accessing the global address of the other Cluster's accumulator. No global memory load/store operation is performed since all data is kept in each L1 local memory. After each iteration, a cluster synchronization barrier is launched before the next iteration. After the weight averaging is completed for a single layer, Cluster 0 loads the weights of the global model with DMA (always shared) and updates them with the gradient tensor computed during the weight accumulation process. Then, it stores the updated weights in the global memory with DMA, waits for the end of the transfer, and starts the weight update process for the next layer.

### 4.2.4 Synchronization

The multi-cluster distributed training process requires a multi-cluster barrier to synchronize all the Clusters when needed. Multi-Cluster Synchronization is managed by introducing specific execution barriers, which are managed by tracking specific events. When a Cluster completes its task, an event with ID=X is launched, and a flag is set to track which Clusters completed the execution. Then, if the Cluster that sent the event is different from Cluster 0, it enters sleep mode. When all Clusters complete the execution, Cluster 0 sends an event to each Cluster to exit sleep mode, and the next task is started.

# 5. Results and Discussion

We analyze the results of both selected applications (i.e., ResNet-18 inference and ResNet-34 training) on the proposed architecture depicted in Fig. 2.1. Tab. 2.3 has the reference for the different configurations. To extract reliable physical implementation information from the architecture, we performed the cluster's physical implementation (down to ready for silicon layout) in 22nm FDX technology from Global Foundries. We used Synopsys Design Compiler for physical synthesis, Cadence Innovus for Place&Route, Siemens Questasim for extracting the value change dump for activity annotation, and Synopsys PrimeTime for power analysis. Area, frequency, and power figures are then scaled to a 5nm tech node more suitable for modern High-Performance Computing (HPC) architectures.

Pros and cons of the proposed architecture configurations have been extensively discussed in the following sections, considering the outcomes of the benchmark's executions. The other configurations have been explored as intermediate results of the design study. In particular, the single-node version is feasible only exploiting wireless, because the routing, in the wired version does not scale with 16 chiplets.

## 5.1 ResNet-18 inference

Firstly, to provide insights into the sources of inefficiency and figure out an upper bound on the performance improvement boosting the communications in such a system and application, we analyze the mapping and latency breakdown of the Resnet-18 inference [9].

### 5.1.1 Initial Consideration

The first source of inefficiency (global mapping) is caused by the fact that not all the clusters are used for mapping network parameters. In our mapping, 322 clusters out of 512 have been exploited. This is an intrinsic characteristic of all systolic architectures exploiting pipelining as a computational model, worsened by the constraints in terms of mapping imposed by IMA. However, this has only an effect on the area efficiency since, in such regular architecture, each cluster can be easily clock and power gated, minimizing the impact on energy efficiency. The second source of inefficiency (local mapping) is caused by the fact that even if a specific cluster is being used, the mapping on it might under-utilize the analog and digital resources. In some cases, parameters cannot fill the whole IMA; in other cases, the array is not used at all. The same happens for digital computing, e.g., in the case of purely digital layers. A possible solution to mitigate this degradation could be to integrate heterogeneous clusters configured to fit better all the possibilities, such as IMA and a single CORE (i.e., analog clusters) or 16 CORES without IMA (i.e., digital clusters).

The third source of inefficiency is caused by the pipeline unbalance. Different layers feature different computational efficiency, where the layer groups are defined depending on the IFM dimension. Some layer groups feature significant area efficiency, thanks to large IFM/OFM, implying high data reuse (i.e., several iterations over the same parameters statically mapped on the IMA).

This study reveals that, unless we would boost communications exploiting wireless features, the main source of inefficiencies, considering this kind of applications and general-purpose architecture AIMC-based architectures, is the non-ideal mapping and the network characteristic. Fig. 5.2 should be intended to understand the breakdown of the inefficiencies in a single chiplet architecture, with all the communications inside the chip. Therefore, the upper bound on this application for the performance is not theoretical but represented by the intra-layer unbalance.

Different considerations should be made for the ResNet-34 training benchmark.

### 5.1.2 Design Space Exploration

We analyze up to six wireless solutions, changing the wireless bandwidth and the number of clusters per chiplet. The wireless configurations have been introduced in Sec. 2.2. The wireless solutions have an incremental wireless exploitation, replacing the off-chip interconnect wired technologies.Fig. 5.3 describes the DSE, figuring out the performance in Tera Operations Per Second (TOPS) and the execution efficiency with respect to reference communications.

In particular, Fig. 5.3A) fixes the wireless bandwidth to 512 Gbit/s, and measures the performance dropping while increasing the number of chiplets. Therefore, by increasing the number of chiplets in the system but fixing the number of clusters, wired solution starts dropping performance while moving to multi-node architecture communicating via PCIe. Anyway, replacing the PCIe with the wireless link (i.e., *wireless 0*), the whole execution does not benefit from it because the nodes communicate with each other with a single-single channel as described in Sec. 2.2. On the other hand, by replacing the HBI interconnects from every communication on the same interposer, the execution time decreases significantly, as we can see from the *wireless 1* bars. The combination of wireless in replaces of HBI and PCIe increases the performance and reduces the drops while scaling down the complexity of the single chiplet.

Fig. 5.3B) fixes the architecture to 16 chiplets, and measures the performance improvement while increasing the wireless bandwidth from 64 Gbit/s to highest obtainable with graphene-based technology, as figured out from this project of 512 Tbit/s. As we can see, what we see is that we actually need that bandwidth on this application, especially for the multi-node configurations proposed.

Fig. 5.3C) shows the above-mentioned trend, fixing the wireless architecture to the *wireless 1*, it measures the efficiency of scaling down the chiplet complexity from 256 to 32 clusters per chiplet. In particular, with 16 chiplets, the efficiency with respect to the 1 chiplet solution is around 63% at 512 Gbit/s and 87% at 8 Tbit/s. The reason for this enormous bandwidth is to demonstrate the trend, where the performance, after 1 Tbit/s is no longer limited by chiplet-to-chiplet communication.This plot shows up that the wireless helps close the performance gap moving from System-on-Chip (SoC) to SiP and beyond to multi-SiP.

Finally, Fig. 5.3D) measures the system's efficiency in both *wired* and *wireless 2* the entire execution time against the ideal communications infrastructure, i.e., no conflicts, no latencies, infinite bandwidths, varying the bandwidth up to about 8 Tbit/s. This enormous bandwidth is to demonstrate that as we have anticipated in Sec. 5.1.1, the bandwidth is a limitation up to 1 Tbit/s. Then, the limitation is no longer on the chiplet-to-chilpet communications. This figure shows also that the execution is still below the upper bound of 46% due to the communications overheads. Comparing against a full on-chip version (i.e., no chip-to-chip communications and with all the data stored in the on-chip memory), the execution efficiency is around 86%, demonstrating the low level of overhead due to off-chip communications thanks to the wireless channels. The overhead with respect the ideal communications is due to on-chip communication conflicts, finite bandwidth, and non-zero latencies that cannot be avoided without significantly increasing area and power consumption.

Fig. 5.4 reports the area efficiency of this application in all the configurations, increasing the number of chiplets in the case of 512 Gbit/s of wireless bandwidth. As we can see, for up to four chiplets, the efficiency is comparable between configurations. This is because the number of I/O hubs and HBMs do not change. On the other hand, the best efficiency when we move to more than four chiplets is always on the multi-node configuration, which has a very low latency execution having private channels even if it is still multiple I/O hub with respect to the single-node configuration. In this last, in fact, the I/O hub area is not dominant with respect to the HBM area and for this reason, the efficiency is still poor. Worth to be noted is that the best efficiency (as for the performance) for single-node configurations is *wireless 5*, which does not have the I/O hub but only single-single channels between every chiplet and memories.

## 5.2 ResNet-34 training

Training benchmark hits our foreseen risk at the beginning of the project, introducing many hours of simulations per test. This is due to the GVSoC simulator, which accurately simulates every instruction on each core. For these reasons, we have used, in this case, a combination of simulated kernel results and analytical models to derive the final results on the entire ResNet-34 training. The kernel executions have extracted the optimized execution of standard 2D convolutions and residual layers, which are the main blocks of the network. Moreover, due to the extreme regularity of the computational model, we have extracted the DMA performance, characterizing the transfers, and we have used these synthetic results to model the entire network and training phase executions. For this evaluation, we have used *wireless 1* architecture configuration, which has been demonstrated as more performant in the previous evaluation.

We also considered the effect of a state-of-the-art tensor-core instead of using the RISC-V cores to execute convolution, linear and residual layers. The latency is per batch. This means that is an average latency per cluster as we have described in Sec. 4.2.

Fig. 5.5 shows the speed-up comparing wired and wireless in both cores and accelerator cases varying the wireless bandwidth. The first consideration is that with low bandwidth, even if the accelerator is 12.5× faster than 16 cores, the communications, which are overlapped with the computation, are dominant in the total amount of cycles. Increasing the bandwidth, as we can see, the speed-up increases and mostly for accelerator because the communication part is more dominant than the case of cores. We reach a speed-up of 6.3× when we use 512 clusters and the accelerator with bandwidth 1 Tbit/s. After that value, as we have already seen in the inference case, the limitation is no longer on the wireless communications but on other communication devices, such as the AXI4 crossbar in the chiplets. With the cores the speed-up is less than with the accelerator because the execution time part in the overall amount of time is bigger and more dominant, but it is still 2.8×.

Fig. 5.6 represents the area efficiency with *wired*, *wireless 1* configurations, with wireless at 512 Gbit/s. the area efficiency decreases, scaling down the number of clusters per chiplet. This is because communication limits the performance while the operations linearly scale up with the number of clusters since the number of clusters coincides with the batch size. This effect is more severe when we go to multi-node configuration (i.e., more than 4 chiplets available) since the communication is slower, concentrating the data movement via a low bandwidth link.

## 5.3 Milestones and final remarks

With our explorations, we have obtained, in the inference benchmark, up to 2.4× and 2.2×, and for the training 3.2× and 3.1× in performance and area efficiency, respectively, considering *wireless 1* configuration and 256 Gbit/s as wireless bandwidth. While considering 512 Gbit/s as wireless bandwidth, we have obtained 3.2× and 3×, and 6.4× and 6.4× respectively in performance and area efficiency in inference and training benchmarks respectively. Tab. 5.1 summarizes the results in terms of performance and efficiency considering *wired* and *wireless 1* configurations, with both bandwidths.

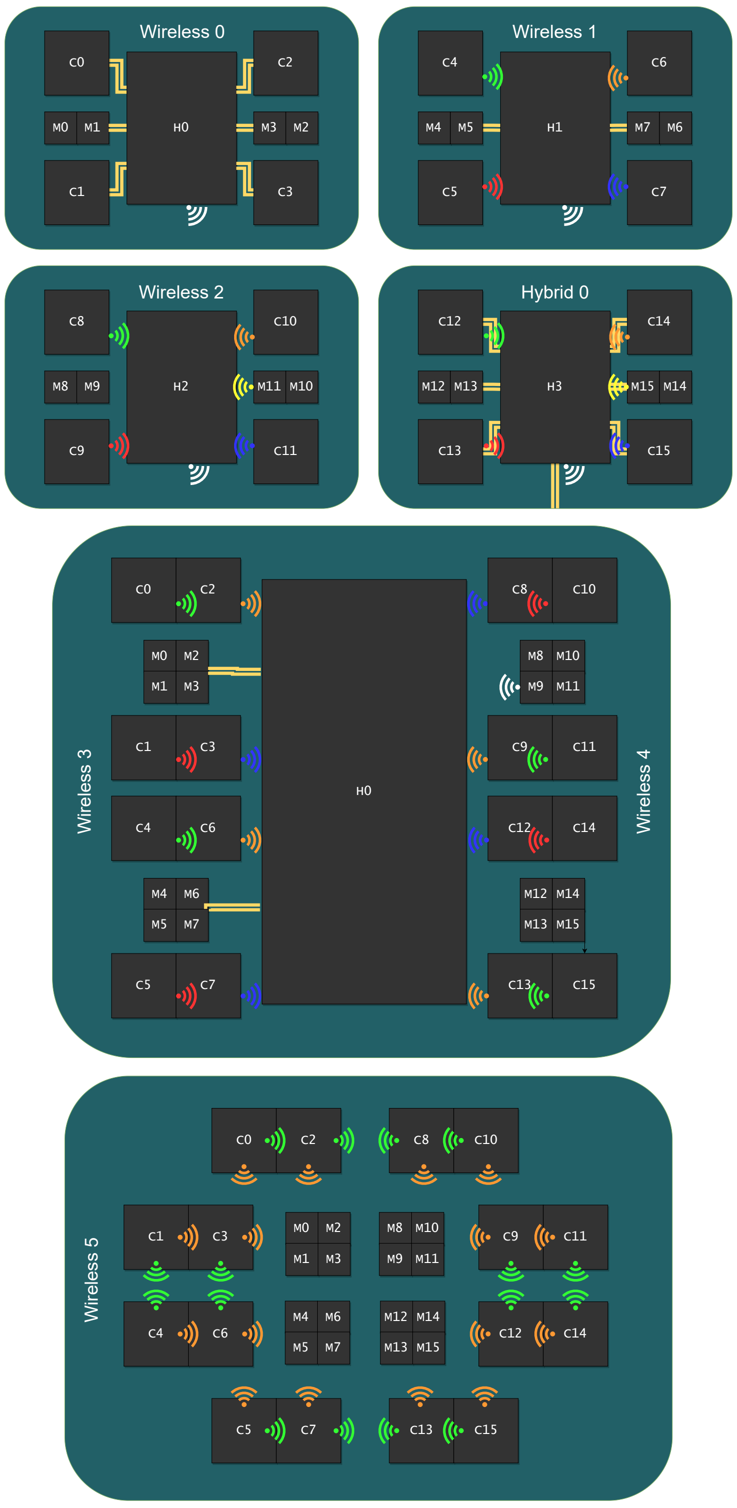


Figure 5.1: Proposed architecture configuration for *wired*, *wireless 0*, *wireless 1*, *wireless 2*, and *hybrid 0* on top. Every color represents a different frequency channel. Proposed architecture configuration for *wireless 3*, *wireless 4*, and *hybrid 1* in the middle. Communications between chiplets are allowed only via I/O hub. Proposed architecture configuration for *wireless 5* on the bottom. Every node has its I/O hub. Communications between chiplets and nodes are allowed only via I/O hub. They are multi-node and single-node architecture, respectively, with up to 16 computing chiplets and 16 HBMs. The configurations details are reported in Tab. 2.3.

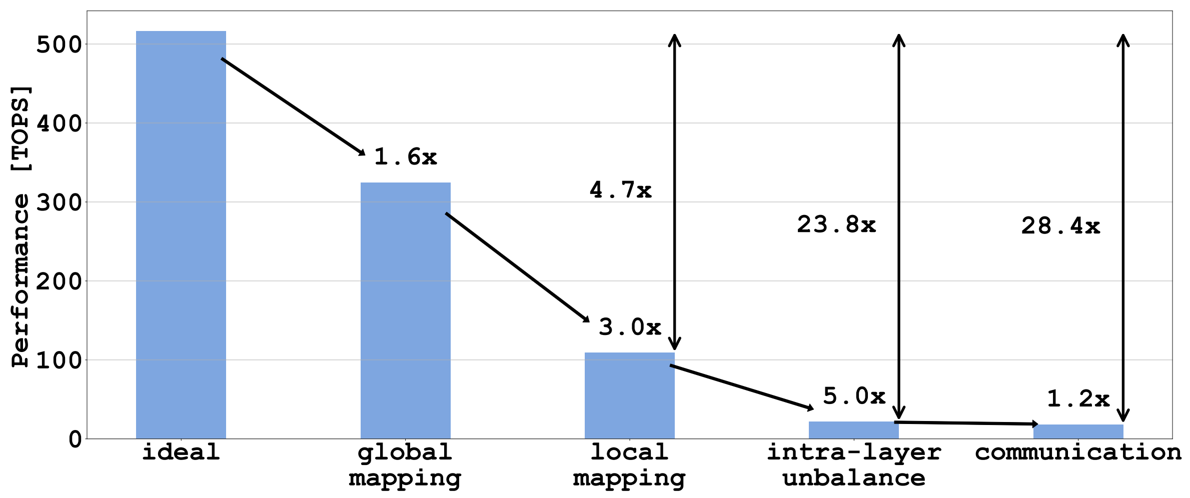


Figure 5.2: Performance degradation considering non-idealities due to static mapping, network topology, and communication.

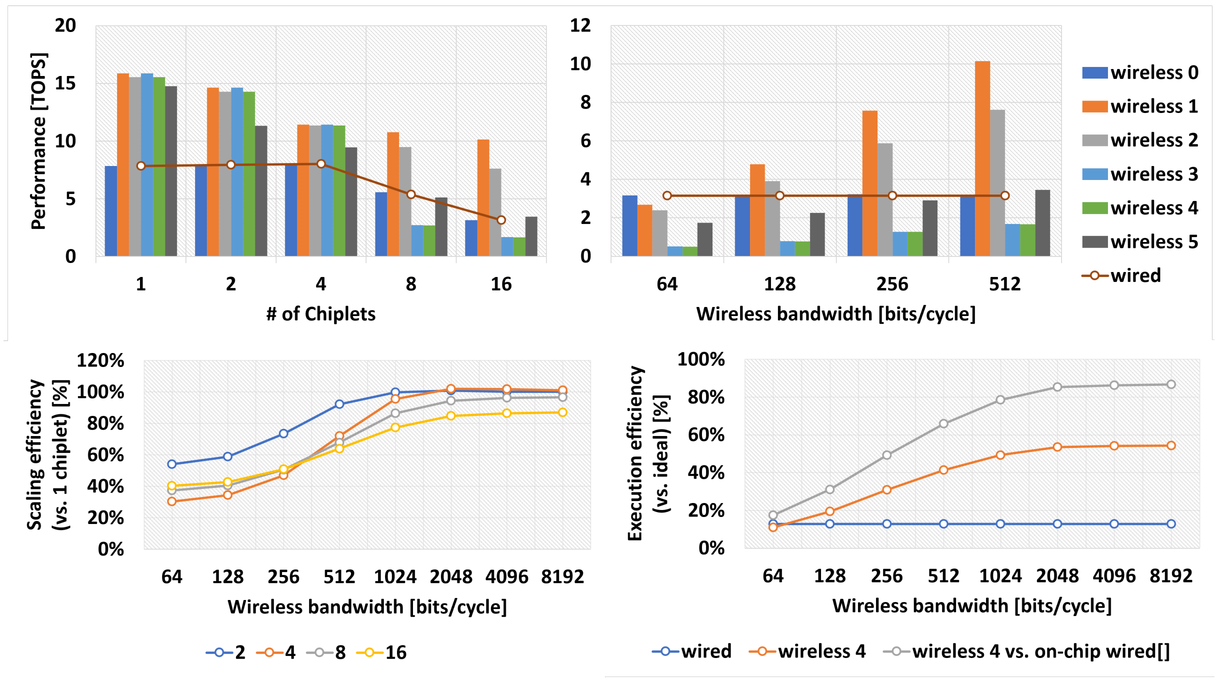


Figure 5.3: ResNet-18 inference results comparing ideal communication, reference wired and several wireless configuration as described in Tab. 2.3.

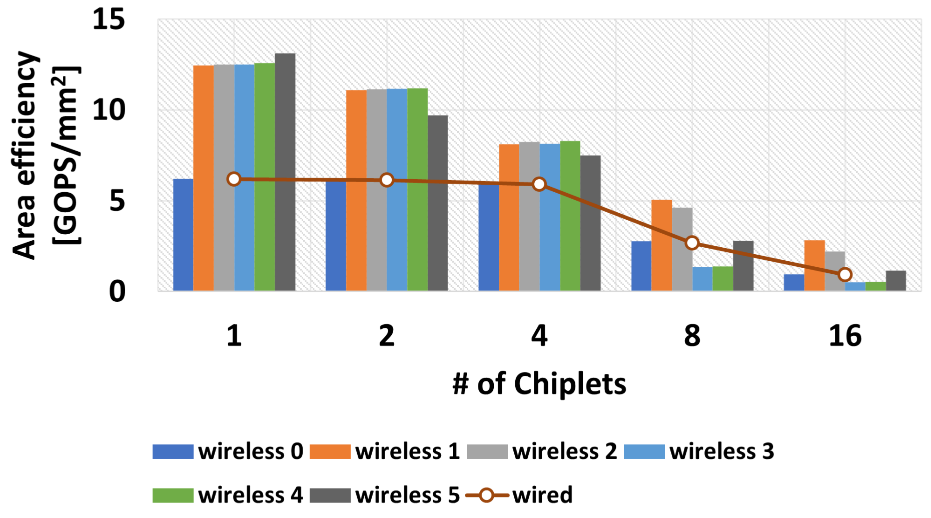


Figure 5.4: Area efficiency of ResNet-18 inference on wired and wireless configurations described in Tab. 2.3.

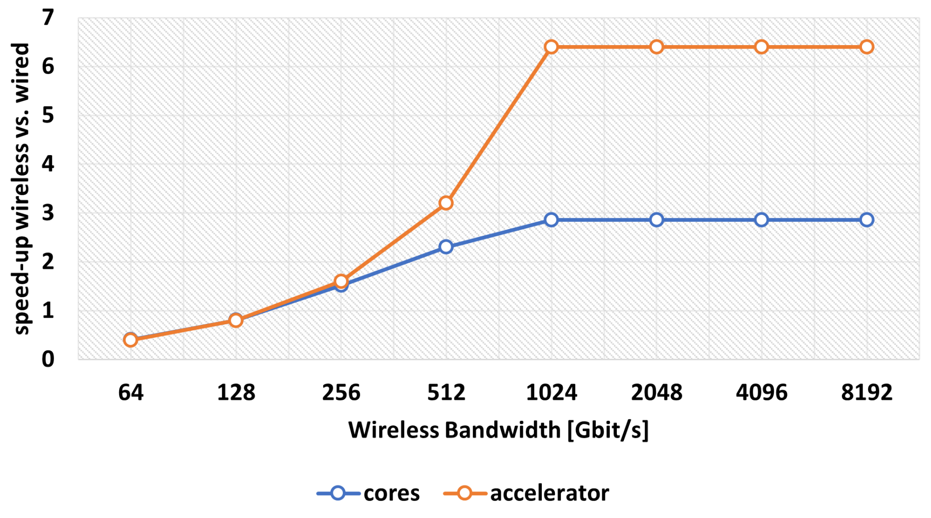


Figure 5.5: Speed-up against wired configuration executing ResNet-34 training in *wireless 1* and with 16 chiplets, considering the 2D convolution, linear and residual layers computed by either RISC-V DSP cores and a state-of-the-art tensor core and varying the wireless bandwidth.

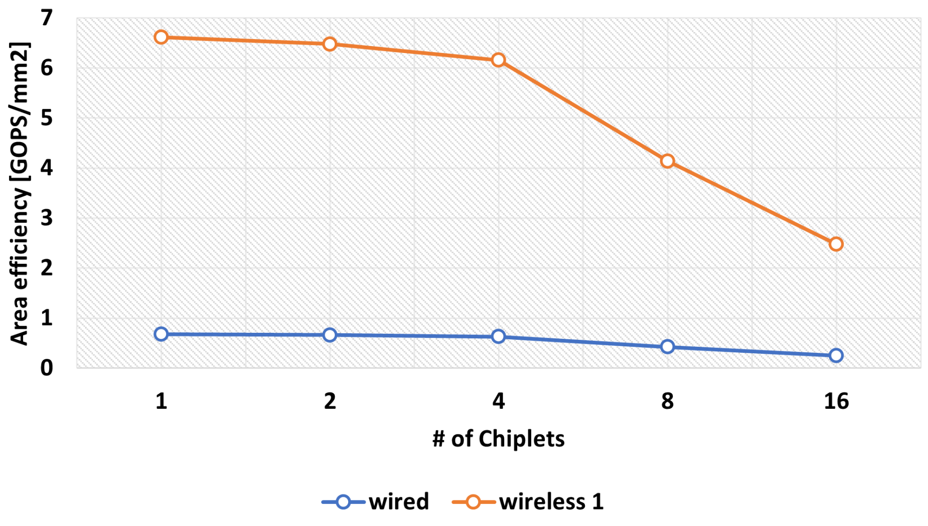


Figure 5.6: Area efficiency of ResNet-34 training on wired and wireless main configuration, with 512 clusters and 512 Gbit/s of wireless bandwidth and considering the 2D convolution, linear and residual layers computed by either RISC-V DSP cores and a state-of-the-art tensor core.

Table 5.1: Results table comparing *wired* and *wireless 1* in both inference and training benchmarks with 16 computing chiplets.

| **Metric** | **Unit** | | **Wired** | | **Wireless 256 Gbit/s** | | **Wireless 512 Tbit/s** | |
|---|---|---|---|---|---|---|---|---|
| | inference | training | inference | training | inference | training | inference | training |
| Technology node | nm | | 5 | | | | | |
| Area | $mm^2$ | | 3380 | | 3371 | | | |
| Performance | TOPS | FLOPS | 3.15 | 1.3 | 7.56 | 4.1 | 10.14 | 8.35 |
| Latency | ms | | 31.02 | 14395 | 12.9 | 4498 | 9.6 | 2250 |
| Area Efficiency | $GOPS/mm^2$ | $FLOPS/mm^2$ | 0.93 | 0.39 | 2.1 | 1.24 | 2.82 | 2.48 |

# 6. Conclusion

We presented an nvAIMC-based multi-tile heterogeneous architecture, analyzing the performance peaks when computing typical CNN workloads and providing insights about the limitations caused by the finite bandwidth of classical communication channels, their rigidity, and the physical size of the analog devices. In this context, the performance and plasticity of emerging on-chip wireless communication paradigms can provide a solid solution in terms of computation performance, especially considering real-life applications with the need of splitting the computation along different nvAIMC.

# Bibliography

[1] S. Pati, S. Aga, M. Islam, N. Jayasena, and M. D. Sinclair, “Computation vs. Communication Scaling for Future Transformers on Future Hardware,” 2023.

[2] S. Naffziger, K. Lepak, M. Paraschou, and M. Subramony, “2.2 AMD Chiplet Architecture for High-Performance Server and Desktop Products,” in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pp. 44–45, 2020.

[3] “AMD Explains the Economics Behind Chiplets for GPUs.” https://www.techpowerup.com/301071/amd-explains-the-economics-behind-chiplets-for-gpus. Accessed: 2023-02.

[4] “Meteor Lake and Arrow Lake Intel Next-Gen 3D Client Architecture Platform with Foveros.” https://hc34.hotchips.org. Accessed: 2023-02.

[5] F. Zaruba, F. Schuiki, and L. Benini, “Manticore: A 4096-core RISC-V Chiplet Architecture for Ultra-efficient Floating-point Computing,” 2020.

[6] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, “Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO ’52, (New York, NY, USA), p. 14–27, Association for Computing Machinery, 2019.

[7] P. Vivet, E. Guthmuller, Y. Thonnart, G. Pillonnet, G. Moritz, I. Miro-Panadès, C. Fuguet, J. Durupt, C. Bernard, D. Varreau, J. Pontes, S. Thuries, D. Coriat, M. Harrand, D. Dutoit, D. Lattard, L. Arnaud, J. Charbonnier, P. Coudrain, A. Garnier, F. Berger, A. Gueugnot, A. Greiner, Q. Meunier, A. Farcy, A. Arriordaz, S. Cheramy, and F. Clermidy, “2.3 A 220GOPS 96-Core Processor with 6 Chiplets 3D-Stacked on an Active Interposer Offering 0.6ns/mm Latency, 3Tb/s/mm2 Inter-Chiplet Interconnects and 156mW/mm2@ 82-Peak-Efficiency DC-DC Converters,” in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pp. 46–48, 2020.

[8] R. Medina, J. Kein, G. Ansaloni, M. Zapater, S. Abadal, E. Alarcón, and D. Atienza, “System-Level Exploration of In-Package Wireless Communication for Multi-Chiplet Platforms,” in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, ASPDAC ’23, (New York, NY, USA), p. 561–566, Association for Computing Machinery, 2023.

[9] N. Bruschi, G. Tagliavini, A. Garofalo, F. Conti, I. Boybat, L. Benini, and D. Rossi, “End-to-End DNN Inference on a Massively Parallel Analog In Memory Computing Architecture,” 2022.

[10] N. Bruschi, G. Tagliavini, F. Conti, S. Abadal, A. Cabellos-Aparicio, E. Alarcón, G. Karunaratne, I. Boybat, L. Benini, and D. Rossi, “Scale up your In-Memory Accelerator: Leveraging Wireless-on-Chip Communication for AIMC-based CNN Inference,” in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 170–173, 2022.

[11] “AMD EPYC Server Processors.” https://www.amd.com/en/processors/epyc-server-cpu-family. Accessed: 2023-02.

[12] “Data Center GPUs for Servers.” https://www.nvidia.com/en-us/data-center/data-center-gpus/. Accessed: 2023-02.

[13] 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.

[14] A. Garofalo *et al.*, “A Heterogeneous In-Memory Computing Cluster for Flexible End-to-End Inference of Real-World Deep Neural Networks,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 12, no. 2, pp. 422–435, 2022.

[15] Y. Tortorella, L. Bertaccini, D. Rossi, L. Benini, and F. Conti, “Redmule: A compact fp16 matrix-multiplication accelerator for adaptive deep learning on risc-v-based ultra-low-power socs,” in *Proceedings of the 2022 Conference amp; Exhibition on Design, Automation amp; Test in Europe*, DATE ’22, (Leuven, BEL), p. 1099–1102, European Design and Automation Association, 2022.

[16] R. Khaddam-Aljameh *et al.*, “HERMES-Core—A 1.59-TOPS/mm2 PCM on 14-nm CMOS In-Memory Compute Core Using 300-ps/LSB Linearized CCO-Based ADCs,” *IEEE Journal of Solid-State Circuits*, vol. 57, no. 4, pp. 1027–1038, 2022.

[17] A. Kurth *et al.*, “An Open-Source Platform for High-Performance Non-Coherent On-Chip Communication,” *IEEE Transactions on Computers*, pp. 1–1, 2021.

[18] “Synopsys High-Bandwidth Interconnect (HBI) PHY IP.” https://www.synopsys.com/dw/ipdir.php?ds=dwc_hbi_phy. Accessed: 2023-02.

[19] “HBM2E.” https://www.micron.com/products/ultra-bandwidth-solutions/hbm2e. Accessed: 2023-02.

[20] “Synopsys Die-to-Die IP Solutions.” https://www.synopsys.com/designware-ip/interface-ip/die-to-die.html. Accessed: 2023-02.

[21] S. Abadal, R. Guirado, H. Taghvaee, A. Jain, E. P. d. Santana, P. H. Bolivar, M. Saeed, R. Negra, Z. Wang, K.-T. Wang, M. C. Lemme, J. Klein, M. Zapater, A. Levisse, D. Atienza, D. Rossi, F. Conti, M. Dazzi, G. Karunaratne, I. Boybat, and A. Sebastian, “Graphene-based Wireless Agile Interconnects for Massive Heterogeneous Multi-chip Processors,” *IEEE Wireless Communications*, pp. 1–8, 2022.

[22] Y. Chen *et al.*, “Channel modeling and characterization for wireless networks-on-chip communications in the millimeter wave and terahertz bands,” *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, vol. 5, no. 1, pp. 30–43, 2019.

[23] R. Guirado *et al.*, “Dataflow-Architecture Co-Design for 2.5-D DNN Accelerators using Wireless Network-on-Package,” in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 806–812, IEEE, 2021.

[24] N. Bruschi *et al.*, “GVSoC: A Highly Configurable, Fast and Accurate Full-Platform Simulator for RISC-V based IoT Processors,” in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pp. 409–416, 2021.