Horizon 2020 Program (2014-2020)

FET-Open Novel ideas for radically new technologies

FETOPEN-01-2018-2019-2020

WiPLASH

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

D4.1: AI Applications Workloads

Contractual Date of Delivery	31/03/2020
Actual Date of Delivery	31/03/2020 (Update: 28/01/2021)
Deliverable Security Class	Public
Editor	Davide Rossi
Contributors	UNIBO (leader), EPFL, IBM, UPC
Quality Assurance	Marina Zapater (EPFL), Albert Cabellos (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	D4.1
Deliverable Responsible	UNIBO
Work Package	WP4
Main Editor	Davide Rossi

Internal Reviewers

1. Marina Zapater
2. Albert Cabellos

Revisions

Version	Date	By	Overview
1.0	09/02/2020	Davide Rossi	First draft
1.1	03/03/2020	Davide Rossi, Marina Zapater, Alexandre Levisse, Sergi Abadal, Irem Boybat	Integrated contributions by EPFL, IBM, UPC.
1.2	06/03/2020	Davide Rossi	First Complete draft
1.3	29/03/2020	Davide Rossi, Marina Zapater, Albert Cabellos	Integrated Internal Reviewers Comments
1.4	30/03/2020	Davide Rossi, Alexandre Levisse	Integrated Additional Contribution by EPFL
1.5	31/03/2020	Sergi Abadal	Final checks, format
1.5.3	16/12/2020	Davide Rossi, Francesco Conti, Giovanni Ansaloni, Sergi Abadal	First pass on implementing feedback from the reviewers
1.5.4	18/12/2020	Davide Rossi	Merged contributions from partners
1.5.5	26/01/2020	Davide Rossi, Francesco Conti	Updated DNN training numbers

Legal Disclaimer

Executive Summary

The main subject of D4.1 is to identify applications workloads to be used in the project as case studies. The workloads (i.e.) relevant kernels extracted from the identified applications will be used within the project for the following purposes: (1) as a baseline on “traditional” state-of-the-art architectures, to highlight bottlenecks, and (2) to evaluate the effect of the proposed THz wireless channels. The presented applications are relevant for a wide variety of real-world domains belonging but not limited to artificial intelligence applications, identified as main driver for the WiPLASH project, and they were selected both for their relevance and for their suitability to be accelerated by THz wireless channels. The deliverable discusses the bandwidth requirement of the selected applications and either estimates their rough potential of acceleration or points out to methods to evaluate such potential. The deliverable will also introduce the hardware architectures that will be evaluated in the project, belonging to three different domains: specialized in-memory computing based hardware accelerators, massively parallel multi-core scalable platforms, and more traditional HPC systems. This deliverable is related to task T4.1: “Heterogeneous System On Chip”.

Abbreviations and Acronyms

SIP	System In Package
PULP	Parallel processing Ultra Low Power platform
SoC	System On Chip
MCU	MicroController Unit
RISC	Reduced Instruction Set Computer
QSPI	Queued Serial Peripheral Interface
I2C	Inter-Integrated Circuit Bus
I2S	Inter-Integrated Circuit Sound
UART	Universal Asynchronous Receiver-Transmitter
DDR	Double Data Rate
DMA	Direct Memory Access
MAC	Multiply And aCcumulate
FPU	Floating-Point Unit
FMAC	Floating Multiply And aCcumulate
HWPE	Hardware Processing Element
AI	Artificial Intellingence
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DNN	Deep Neural Network
SGD	Stochastic Gradient Descent
GPU	Graphic Processing Unit
HPC	High-Performance Computing
DLRM	Deep Learning Recommendation Model
FC	Fully Connected
CTR	Click Through Rate
BFP	Banach Fixed Point
ReLU	Rectified Linear Unit
FL	Federated Learning
GNN	Graph Neural Network
RNN	Recurrent Neural Network
PE	Processing Element
KPI	Key Performance Indicator
DNA	Deoxyribo Nucleic Acid
NGS	Next-generation sequencing
BWT	Burrows-Wheeler Transform

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

Table of Contents

List of Figures

List of Tables

1 Introduction

Recent years have seen the emergence of heterogeneous architectures integrating multiple general-purpose, specialized computing units, and memories. This diversification trend comes together with the extreme parallelism, with energy being the main driver for such architectures, an objective that cannot be met anymore with the traditional scaling-driven optimization cycles that have maintained Moore’s Law trend for decades. The diminishing returns of transistor scaling, the severe scalability issues of conventional memory–interconnect architectures of heterogeneous massively parallel processors, and the cost of manufacturing large chips are causes of this paradigm shift.

Hardware specialization opens a huge range of possibilities from the architecture perspective, but also urgently calls for key enablers at the integration and interconnection stages. The use of System-in-Package (SiP) and chiplet systems based on 2.5D stacking on silicon interposer, in embedded and high-performance environments, are the most common alternatives, often employing carefully designed variants of a wired Network-on-Chip (NoC) to transport data between the different components. However, as we move off-chip, pin constraints limit the bandwidth and flexibility of the available communication schemes, narrowing down the applicability and hindering the scalability of heterogeneous systems. In this context, exploiting the wireless plasticity pioneered by the WiPLASH project, thanks to the implementation of wireless communication at the functional unit level, is expected to lead to a radical step further in computing by designing a new breed of massive heterogeneous architectures at extreme scales.

This deliverable aims at providing an overview of the application workloads identified by the consortium during the first six month of project, and mapping these applications to the three different domains envisioned. The deliverable will first provide an initial specification of the architectures targeted by the WiPLASH project, in the three different domains: 1) in-memory computing based architectures, 2) massively parallel heterogeneous programmable systems 3) HPC systems. Furthermore, it will describe the identified applications, their relevance for the scientific and industrial community, and their suitability to be accelerated by the architectures developed in WiPLASH. We also outline the bandwidth requirements of these applications and the potential acceleration offered by the sheer increase of bandwidth offered by addition of the wireless links.

The reminder of the document is organized as follows. Section 2 will provide an introduction to the WiPLASH architectures, Section 3 will describe the proposed applications and their workloads relevant to the WiPLASH architectures, Section 4 will provide some final remarks.

2 Target Architectures

This section provides a brief introduction of the architectures explored within WiPLASH, with the goal of understanding the suitability of the domain of applications to architectures targeted.

2.1 In Memory Accelerators-Centric Architectures

In-memory computing is an emerging computing paradigm that could enable the execution of various applications, such as deep learning inference, at extremely high energy efficiency and throughput. Among the others, one interesting approach based on in-memory computing is based on memory arrays computing fundamental operations such as dot products in the analog domain with extremely high energy efficiency. The fundamental unit of in-memory computing is a set of memory devices organized in a crossbar array, as depicted in Figure 1. A matrix-vector multiplication can be executed in constant time by storing the matrix elements as conductances of the memory devices and applying the vector values as voltage levels on the word lines. Owing to Ohm's law and Kirchhoff's current laws, the result of the matrix-vector multiplication is obtained as currents on the bit lines. Note that both conventional charge-based memory devices and emerging resistive-based devices are good candidates for in-memory computing.

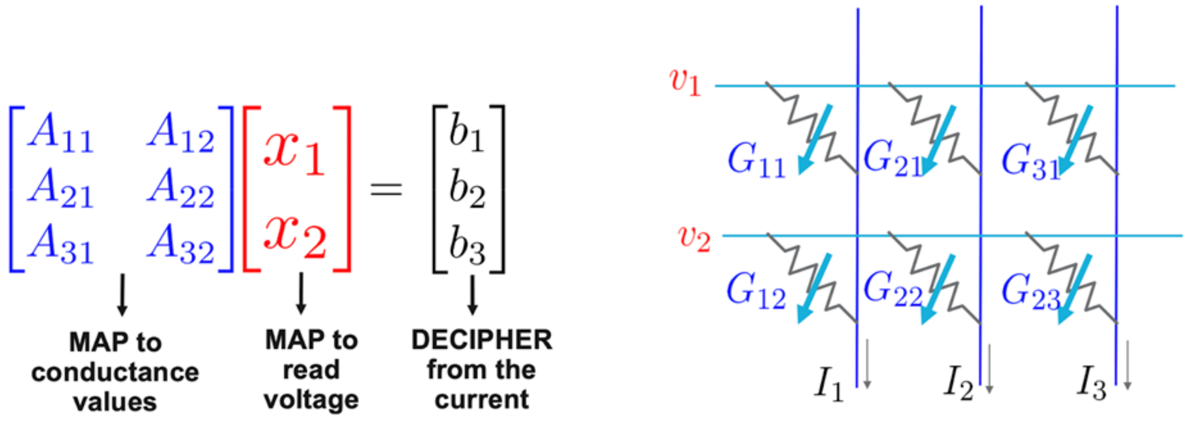
$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$
MAP to conductance values
MAP to read voltage
DECIPHER from the current
v_1
v_2
G_{11}
G_{21}
G_{31}
G_{12}
G_{22}
G_{23}
I_1
I_2
I_3

Figure 1. In-memory computing unit.

A computational memory core can be defined as a unit composed of the memory crossbar array, the digital processor, and input and output memories. A multitude of such interconnected cores composes the in-memory computing-based architecture. During execution, the output of a computational memory core is passed as an input to another computational memory core, as shown in Figure 2. Hence, the data is transferred from one core to the next in a pipelined fashion. In the execution of a dataflow on the in-memory computing-based architecture, the communication fabric is of significant importance, allowing the pipeline to run unstaggered and ensuring maximum throughput. The in-memory computing-based architecture can leverage on the constant time complexity of the matrix-vector multiplication to execute with unprecedented performance deep learning tasks such as image classification and natural language processing.

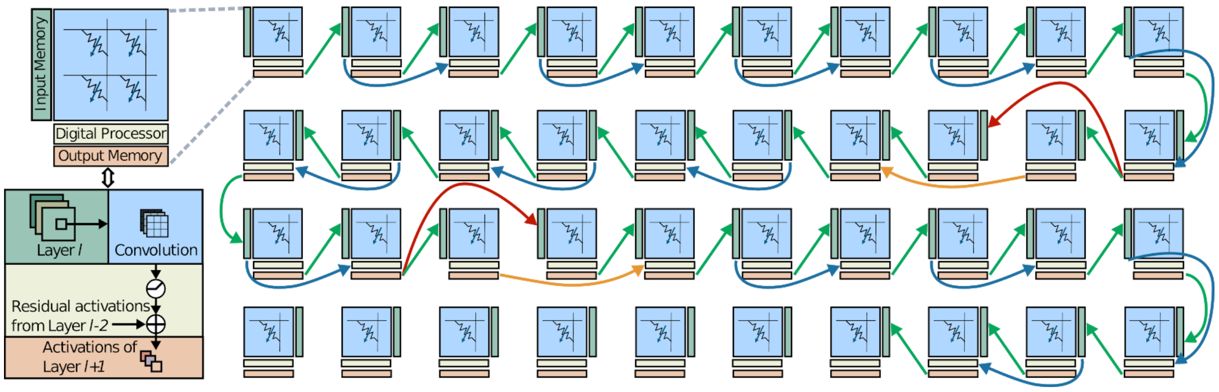
Input Memory
Digital Processor
Output Memory
Layer *l*
Convolution
Residual activations from Layer *l-2*
Activations of Layer *l+1*

Figure 2. In-memory computing-based architecture composed of computational memory cores.

For a sample application of deep learning inference (see Section 3.1.1 for further details), the in-memory computing-based architecture results in the design parameters shown in Table 1.

Metric	**Value**
Number of cores	34
Data rate (per core)	5 Gbps
Size of core	576x576
Energy consumption per core	133 nJ
Degree of connectivity	Max. 3 channels (3 interconnected cores)
Reconfigurability	Desired

Table 1. Design parameters for the application of deep learning inference on in-memory computing-based architecture. ResNet-32 network for the task of CIFAR-10 image classification is taken as a reference. The crossbar array is assumed to be composed of phase-change memory, fabricated in 90 nm technology node.

2.2 Low-power computing platforms for massively parallel processing

In order to improve the programmability of computing platform with respect to accelerator-centric in-memory computing matrixes, in WiPLASH we leverage the architectural template of the open-source PULP platform. PULP (Parallel processing Ultra Low Power platform) is a design-configurable, scalable clustered many-core computing platform, written in synthesizable System Verilog, that allows to define at design-time its configuration with as many computing clusters as the applications require. Hence the same architectural template can be used both in the context of low-end applications and as general-purpose accelerator for high performance computing systems.

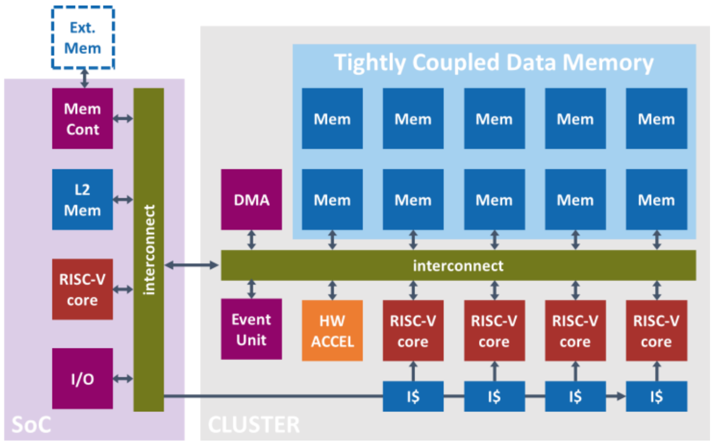

Figure 3. PULP SoC Architecture.

PULP SoCs [1] are built around an advanced MCU controlled by a 32-bit RISC-V processor and a full set of peripherals typical of low-power microcontrollers, as shown in Figure 3. It includes all the peripherals typical of low-power microcontrollers such as QSPI, I2C, I2S, parallel camera interface and UART. While in the current instance of PULP platform on-chip memory is extended via an up to 32 Mbytes DDR HyperBus interface (800 Mb/s), which is the peripheral providing highest performance with wired communication, for the low-end performance embodiment (MCU like). Data transfers from/to the peripherals are managed by a multi-channel I/O DMA (μDMA) to minimize the amount of interactions and the workload of the controlling core when performing IO. In the high-end embodiments of the platform, the peripheral subsystem can be replaced by models of high-performance memory controllers (e.g. DDR4). WiPLASH will explore mode advanced IO interfaces based on THz wireless channels.

When the computational requirements of applications cannot be satisfied by the controlling processor. The cluster, residing on a dedicated voltage and frequency domain, can be turned on and adjusted to the right voltage and frequency. It contains a parametric number of RISC-V cores supporting the RVC32IM instruction set [2], plus extensions targeting energy-efficient digital signal processing such as hardware loops, load/store with pre/post increment, multiply and accumulate (MAC) vector instructions (RVC32IMFX) [3]. A parametric number of floating-point units (FPU) can also be shared among the processors of the cluster, implementing common floating point operations including FMAC, a key operation for near sensor tasks such as filtering and neural networks. A multi-ported and multi-banked tightly coupled data memory is accessible in just one cycle by all processors and hardware assistance for synchronization are also provided.

A DMA engine optimized for multi-core clusters ensure that data can be moved in and out the cluster with the necessary efficiency, although DMA configuration is the direct responsibility of the programmer (i.e. the data memory is not a cache). Dedicated shared memory hardware processing elements (HWPE) can be added to the cluster to improve performance and energy efficiency whenever application requirements cannot be satisfied by purely parallel software computation [4]. One possible customization of the PULP clusters in WiPLASH leads to the exploitation of PULP clusters has heterogeneous, wireless THz channels augmented, in-memory computing engines coupling programmable processors with the accelerators described in previous section, as depicted in Figure 4.

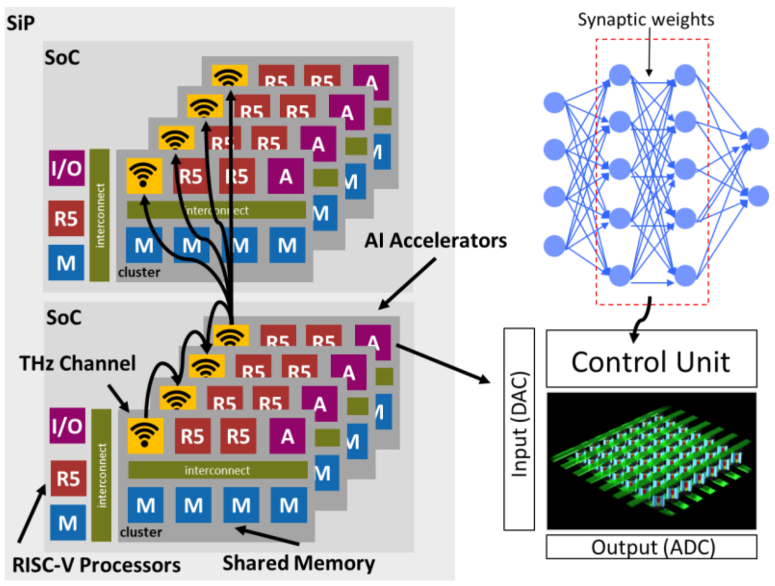

Figure 4. Possible Embodiment of PULP SoC augmented with Wireless THz Channels and In-Memory computing cores.

2.3 HPC Architectures

HPC architectures are comprising one or several clusters of high-performance multi-core systems, either RISC-V or ARM, connected to in-memory accelerators. Such complex and heterogeneous system cannot be emulated PULP VirtualPlatform, but can be simulated with gem5-X [5], running a full Linux OS distribution (as a result of the work performed in WP5).

Gem5-X is an extended version developed in EPFL of the gem5 architectural simulator. Simulation with gem5-X can consider in-order cores, out-of-order cores or any combination of both (it can support up to 64 in-order cores in a single simulation). Architectural extensions can be introduced and their effects assessed on real applications running. For example, gem5-X supports High Bandwidth Memories (HBM) and in-cache computing or enables the introduction of new instructions. Such a simulation model is validated and fine-tuned against characterizations against RISC-V, ARM and x86 server platforms. Gem5-X simulations calibrated for a ARMv8 A53 in-order core found on the ARM Juno development board, and running an Ubuntu 18.04 LTS software environment demonstrates less than 4% timing inaccuracy on profiling tests compared to physical hardware.

HPC architectures exhibit higher thermal dissipation than low power architectures, thereby coupling with thermal simulations can be required to ensure the correct functionality of the proposed architecture. Power traces can be extracted from the gem5-X simulator and fed into the thermal simulator 3D-ICE [6] to determine associated thermal dissipation. 3D-ICE simulations are compared against real measurements and show off less than 7% error.

Considering the previously introduced architectures and simulation platform, wireless interconnect and protocols can be implemented between clusters, cores, or between clusters/cores and accelerators. Such a simulation framework enables to determine, for a given application and a set of constraints, the most optimized architecture for a power/performance/thermal target.

3 Applications

This section provides an overview of the applications proposed in WiPLASH, starting from more traditional convolutional neural networks inference and training, to emerging deep learning applications such as DLRM systems and other applications relevant to the scientific and industrial communities not belonging to AI.

The applications are summarized in Table 2, which provides also some of the KPI ranges expected for the applications and the mapping of the applications on the target architectures. The table also distinguishes between applications that will be evaluated in specialized hardware-based architectures and deeply embedded platforms, from those that would be evaluated in high-performance systems with a more complex architecture. The distinction is necessary because the effort required to implement and optimize applications on specialized hardware-based and deeply embedded platforms, such as array-based architectures and deeply embedded multi-core systems with explicitly managed memory hierarchy, is much higher than the effort required for high-performance systems with flat memory hierarchy and operating systems [7].

For the reasons above, the studies related to hardware-based and deeply embedded architectures is expected to be restricted to the well-known application domains of inference and training of embedded deep neural networks such as MobileNets. These applications have relatively small memory footprints (up to 8MB for inference and up to 64 MB for training), still available in DRAM memories for deeply embedded applications such as Cypress HyperRAM [8], and APMEM [9]. These applications are still expected to take advantage of the high bandwidth (orders of magnitude larger than SoA devices), low-latency and reconfiguration capabilities of THz wireless channels due to the extremely limited on-chip memory available in deeply embedded devices (typically up to 1MB for embedded AI processors) [10].

The peak efficiency expected for these systems is in the range of TOPS/W, thanks to the exploitation of embedded in-memory computing based accelerators. Inference problems more suitable to be accelerated by this kind of units. On the other hand, training problems also require both inference and backward passes, also requiring floating-point (high-precision) computations degrading on average the overall energy efficiency of the application. The other emerging applications are more suitable for high-performance computing architectures, featuring larger memory footprints and a peak efficiency expected to be in the range of hundreds of GOPS.

In order to evaluate the potential benefits of augmenting the described platforms with THz wireless communication capabilities, we have computed the bandwidth requirements for the different applications. Bandwidth requirements have to be intended as the bandwidth needed each application in order to be 100% compute bound (in other words, 0% communication bound). The bandwidth is provided as a range as for the application performance: the larger is the performance expected for each application (GOPS), the larger the bandwidth necessary to sustain this performance will be. However, aligned with the considerations above, we again note that the bandwidth improvements would lead to predictable performance gains ONLY in the case of specialized architectures. On the other hand, in complex HPC architectures, increasing the interconnect bandwidth does not necessarily directly correlate with predictable speedups due to the effect of the memory hierarchy or the operating system. In that case, estimating the gains of the wireless technology will require full system simulations.

Application	Leader	Memory Footprint [MB]	Bandwidth requirement [Mb/s]	Performance [GOPS]	Power Envelope [mW]	Efficiency [GOPS/W]	GP Computing Platforms	Specialized PULP Platform	Accelerator-Centric Platform
Embedded DNN Inference	IBM	1-8	15-750	1-50	10-3K	500-5K		✓	✓
Embedded DNN Training	UNIBO	8-64	240-12K	1-50	10-3K	100-1K		✓	✓
DLRM	EPFL	1K-20K	8K-1M	100K	100K-400K	250-1K	✓		
Federated Learning	EPFL	1-100	4K-40K	1-500	1K-10K	100-1K	✓		
Next-generation sequencing	EPFL	1K-4K	15K-150K	10K-50K	50K-200K	200-250	✓		
Real-time video surveillance	EPFL	10-4K	50K-150K	10-50	5K-10K	1-5	✓	✓	
Graph Neural Networks	UPC	10-10K	1K-1M	50-8K	2K-300K	100-700	✓	✓	

Table 2. WiPLASH Applications summary and main features.

3.1 Embedded CNN Inference

Lately, Artificial Neural Networks (ANNs) have been extensively used in regular data domains. Specifically, Convolutional Neural Networks (CNNs or ConvNets) have been exploited for images, video, sound recordings, etc., which has consequently allowed us to perform relevant tasks over them such as image classification [19], object detection [20] and pose estimation [21] among others. Most ConvNets are built from the same basic building blocks: convolution layers, activation layers, and pooling layers. One sequence of convolution, activation, and pooling is considered a stage, and modern deep networks often consist of multiple stages.

In-memory computing-based architectures can leverage on the constant time complexity of the matrix-vector multiplication to execute with unprecedented performance convolutional layers of deep learning tasks. In the in-memory computing-based architecture, the dataflow occurs as such: the layers of the CNN are each mapped to the cores. The execution occurs in a pipelined fashion across the dot products: the core assigned to the l-th layer holds in its input memory the activations calculated and transmitted by the core assigned to the (l-1)-th layer. When enough input activations are present in its input memory, one core calculates one dot product, resulting in the output activations from a pixel position across all channels. These activations are then transmitted to the core executing the subsequent layer.

Figure 5 shows an example of the dataflow (color-coded for different operations/components), with the dot product being mapped on the computational memory array.

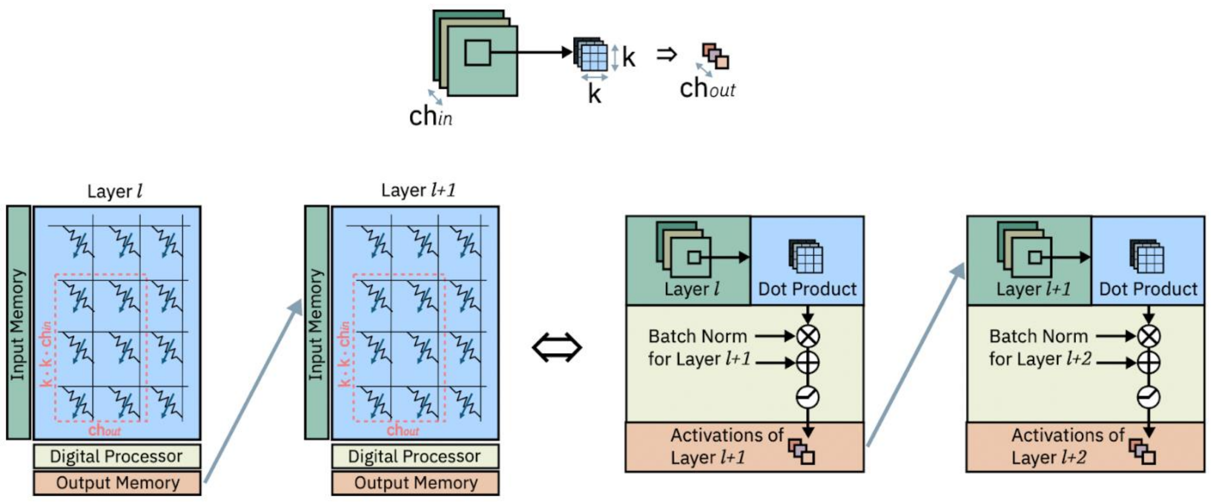

Figure 5. Dataflow execution of convolutional neural network layers in in-memory computing devices.

While results are already been on display for physical communication links between cores, on-chip wireless communication opens up new possibilities for a plastic communication fabric across the chip, as shown in Figure 6.

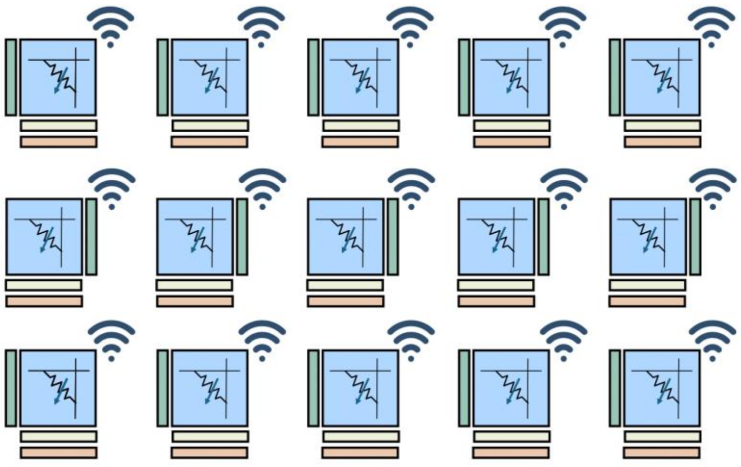

Figure 6. On-chip wireless communication for an in-memory computing-based architecture can allow new, plastic communication fabrics for inter-core communication.

On the other hand, since the in-memory computing-based architectures can only provide matrix-multiplication computing, some functionalities such as activation and pooling layers still have to be computed in the digital platform, opening new frontiers for massively parallel architectures composed of PULP clusters coupled with in-memory computing accelerators with much more flexibility than custom architectures.

To estimate the bandwidth consumed by a typical embedded inference application, we used a *MobileNet v2* network as benchmark. We estimated the bandwidth consumption as being related to weight access; using 4-bit per weights, and targeting full-throughput of 50 GOPS, the maximum bandwidth requirement is 768 Mb/s. Assuming a PULP system featuring a 1.6 Gb/s Cypress HyperBUS [8] as external interface, although wireless communication would not improve performance on this specific use case, we could expect an energy improvement of around 5x when moving from HyperBUS (110 pJ/bit) to THz wireless channels (assuming 1 pJ/bit). Beyond that, it is important to note that the improvements of wireless connectivity do not stop at increasing bandwidth or efficiency. In the in-memory computing case shown above, the reconfigurable connectivity allows the accelerator to be able to adapt to a myriad of different CNNs while minimizing overprovisioning of the network-in-package. We aim to evaluate this non-trivial improvement achievable in this case later in the project.

3.2 Embedded CNN Training

Modern Deep Neural Networks (DNNs) have to be trained on clusters of GPUs and millions of sample images to be competitive. Complex networks can take weeks to converge during which the involved compute machinery consumes mega joules of energy to perform the exa-scale amount of operations required. Inference, i.e., evaluating a network for a given input, provides many knobs for tuning and optimization. Substantial research has been performed in this direction and many good hardware accelerators have been proposed to improve inference speed and energy efficiency [11]. The training of DNNs is much harder to do and many of these optimizations do no longer apply. Stochastic Gradient Descent (SGD) is the standard algorithm used to train such deep networks [12].

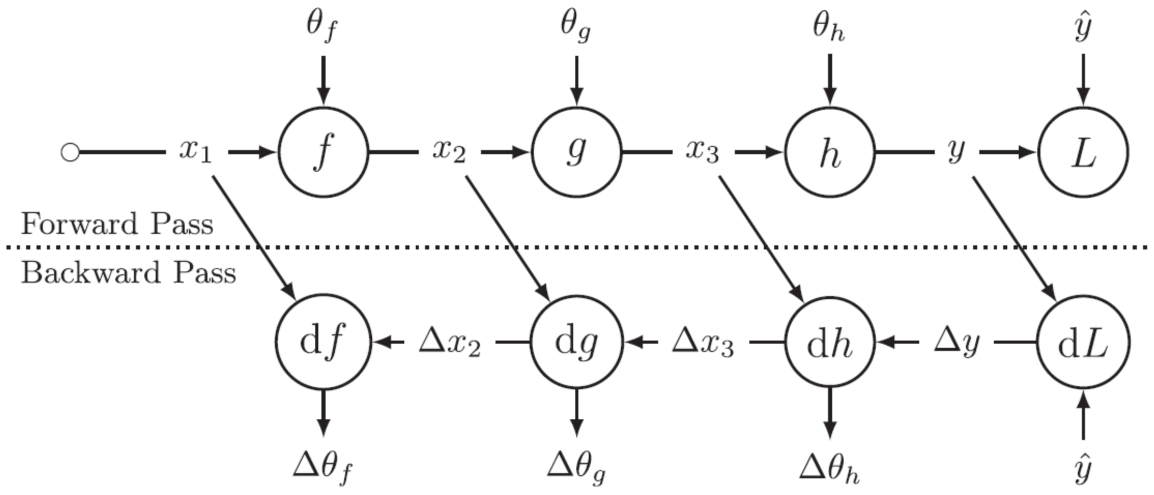

Figure 7. Data dependency graph of the forward pass (above) and backward pass (below).

Consider Figure 7 which shows the data dependencies when training a simple neural network. While inference is concerned only with finding y, training aims at finding the gradients (D_u) which introduces a data dependency that requires us to temporarily store the output x_i; y of every layer. This also prevents optimizations such as fusing activation or sub-sampling functions with the preceding layer, putting an extreme pressure on the memory systems of architectures used for training. Moreover, while it has been shown that inference is robust to lowering arithmetic precision [11], the impact of fixed-point or reduced-precision floating-point (FP) arithmetic on training is not yet fully understood, somehow limiting memory footprint minimization techniques such as extreme quantization. Until additional light is shed on the topic, a training accelerator must support 32 bit FP arithmetic to be able to compete with the ubiquitous GPU.

In this scenario, while most traditional architectures for training of deep neural networks belongs to the HPC domain, adaptation of DNNs to new tasks and functionalities in an embedded domain is considered an open research problem. Techniques based on one-shot and few-shot learning [13][14] tackle the problem by dividing the network in a feature-extractor, which is trained entirely offline to derive a metric embedding of the input, and a final stage clustering the various classes depending on their respective distance. A more complete solution to the problem is that proposed by continual learning [15][16] by modifying the loss and training algorithms in order to retrain a few layers of the network for a new task without forgetting the previous ones. There is significant interest in these technologies, particularly for the purpose of integration in robotic devices, however several problems (e.g., significant memory footprint) have to be solved before a full continual learning approach can be deployed on an embedded device such as PULP, potentially leverage THz channels for communication with in-package memories.

Another scenario where learning has to be integrated into the embedded devices is training of in-memory computing elements, in order to reduce negative impact of process variations between the in-memory computing cores used for DNN inference. This heterogeneous integration between the in-memory computing cores and the "high-precision" digital unit requiring floating-point support poses an extreme challenge on the communication interface between these two subsystems, that can be solved exploiting wireless THz channels developed in WiPLASH.

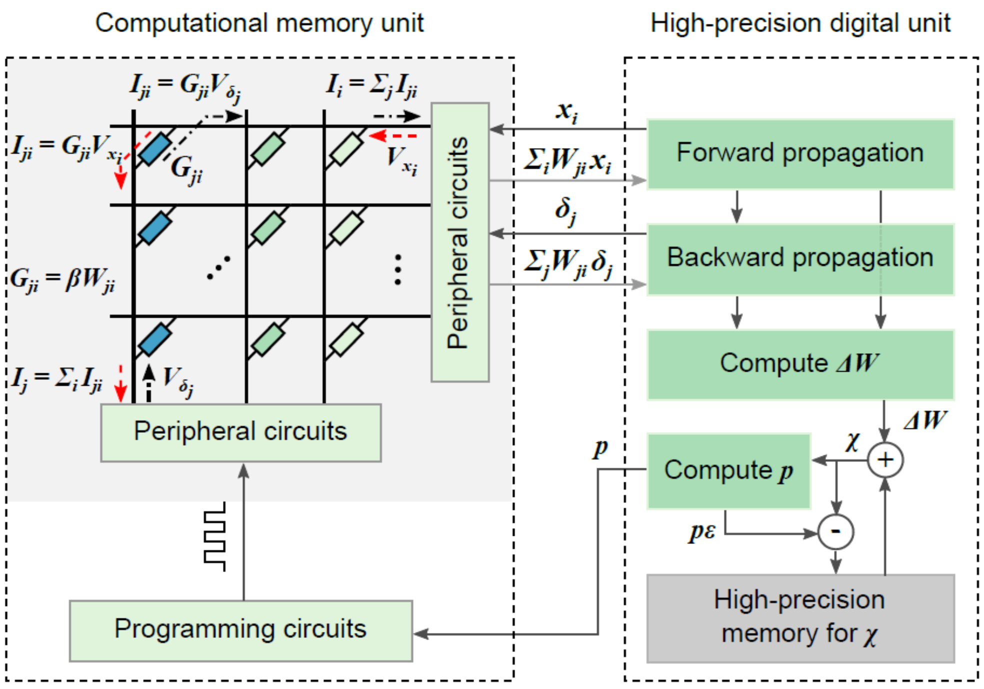

Figure 8. Mixed-precision computational memory architecture for deep learning.

To estimate the bandwidth consumed by an embedded learning application, we used a *MobileNet v2* network as benchmark. We estimated the bandwidth consumption as being related to weight access in both directions (read for forward propagation, write for backward propagation); using 32-bit per weight, and targeting full-throughput of 50 GOPS, the maximum bandwidth requirement is 12.3 Gb/s. The introduction in the system of THz wireless channels for communication with external memory would improve performance of the application by 7.6x over a system employing 1.6 Gb/s

HyperBUS interface [8], assuming a peak bandwidth of 100 Gb/s for THz wireless channels. Moreover, assuming a transfer efficiency of 1 pJ/bit, THz wireless channels would improve the energy efficiency of the application by more than one order of magnitude.

3.3 Recommendation Systems

Personalized recommendation is the task of recommending new content to users based on their preferences. In other words, deep learning-based recommendation systems are used throughout industry to predict rankings for news feed posts and entertainment content. Estimates show that up to 75% of movies watched on Netflix and 60% of videos consumed on YouTube are based on suggestions from their recommendation systems. Moreover, in 2018, McKinsey and Tech Emergence estimated that recommendation systems were responsible for driving up to 35% of Amazon's revenue [17].

The main task of such services is the to accurately, and efficiently rank content based on users' preferences and previous interactions (e.g., clicks on social media posts, ratings, purchases). Building highly accurate personalized recommendation system poses unique challenges as user preferences and past interactions with content are represented as both dense and sparse features. For instance, in the case of ranking videos, there may be thousands of potential videos that have been seen by millions of viewers. However, individual users interact with only a limited number of videos. This means interactions between users and videos are sparse. Sparse features not only make training more challenging but also require intrinsically different operations.

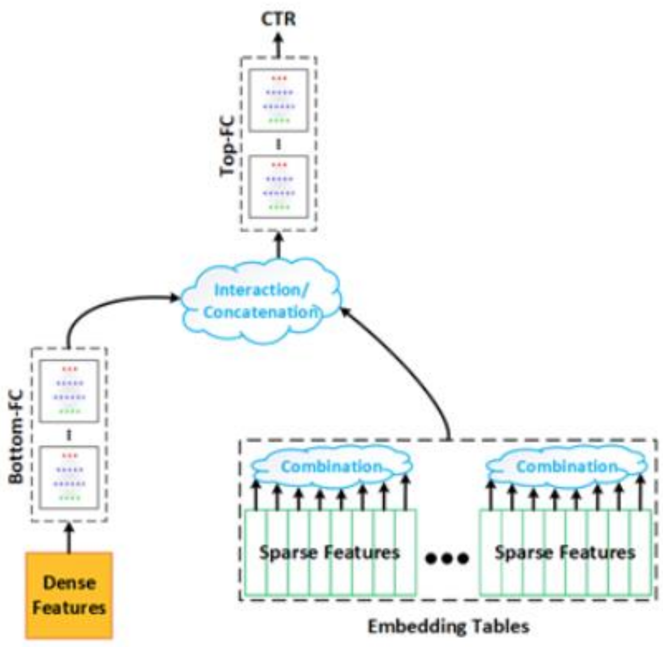

Figure 9. Simplified DLRM architecture.

Figure 9 shows a simplified architecture of state-of-the-art DNNs for personalized recommendation models used at Facebook. This so-called Deep Learning Recommendation Model (DLRM) consists of a variety of operations such as Fully

Connected (FC) layers, embedding, pooling, and non-linearities, such as ReLU. At a high-level, dense and sparse input features are separately transformed using FC layers and embedding tables respectively. The outputs of these transformations are then combined and processed by a final set of FC layers. The inputs, for a single user and single post, to recommendation models are a set of dense and sparse features. The output is the predicted click-through-rate (CTR) of the user and post. Dense features are first processed by a series of FC layers, shown as the Bottom-FCs in Figure 9. On the other hand, sparse input features, represented as multiple vectors of sparse IDs, must first be made dense. Although each vector of sparse feature can be transformed to dense vectors using FC layers, its compute demands would be significant. Therefore, embedding tables are used. Each vector is paired with an embedding table, and each sparse ID is used to look-up a unique row in the embedding table. The rows of the embedding are then combined into a single vector. Finally, these vectors and the output of the Bottom-FC layers are concatenated, and processed by the Top-FC layers.

A key distinguishing feature of DNNs for recommendation systems, compared to CNNs and Recurrent Neural Networks (RNNs), is the use of embedding tables. Embedding tables are used to transform sparse input features to dense ones. The dense representations are subsequently processed by a series of more traditional layers including, FC, pooling, and ReLU non-linearities.

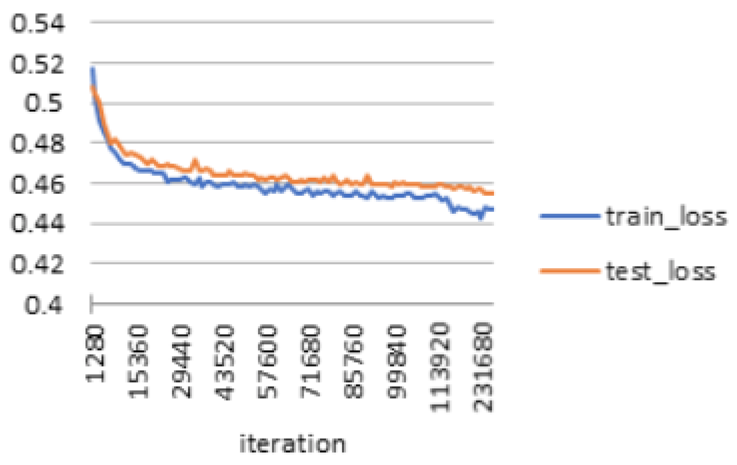

Figure 10. Train loss vs test loss with default DLRM parameters.

DLRM is provided as open-source by Facebook implemented through PyTorch. This implementation runs with default values for different parameters, such as shape of the embedding, shape of the bottom and top MLP, activation functions, loss threshold, number of workers, sparse feature size, etc. The training loss vs. the test loss is shown in Figure 10 for the default parameters. For each specific application the optimal values of these parameters can vary. As the first step of profiling this DLRM implementation, we run it through different number of workers and batch sizes, while using Kaggle dataset. As shown in Table 3, these two parameters affect training time and GPU utilization.

Table 3. Training time, test loss, and GPU utilization with different number of workers and batch sizes.

Batch Size	Num. Workers	Time (s)	test_loss	GPU Utilization (%)
512	0	7.53	0.4588	2.11
512	2	9.54	0.4589	2.89
512	8	9.87	0.4587	3.28
128	0	7.73	0.4568	4.2
128	2	8.57	0.4569	3.6
128	8	8.85	0.4566	2.9
32	0	7.29	0.4558	5.95
32	2	9.09	0.4556	4.22
32	8	9.33	0.4556	2.32

The most suitable architecture for this application seems to be Architecture #3, as this application has HPC requirements. Even in this context, sustaining communication requirements is a major challenge for state of the art (wired) interconnects. While such requirements vary across implementation and deployments, depending on a number of factors including batch size and architectural hyperparameters, they are consistently very high. As examples, we measured a peak bandwidth exceeding 1GB/s in our experiments targeting a Nvidia V100 GPUs, while Facebook indicates that, in their setup, DLRM is only completely compute-bound if the available bandwidth exceeds 1TB/s.

In this case, it is clear that the wireless interconnect will hardly achieve the bandwidth required to make this task completely computation-driven. In fact, it is of the architects' best interest to explore how to best exploit the characteristics of the wireless interconnect with its limited bandwidth. For instance, besides using the wireless interconnect to accelerate the partial DNN runs, the low-latency broadcast can be used to distribute and gather tasks from the workers or fetch values from a centralized embedding table. However, as indicated previously, the complexity of HPC platforms makes it difficult to make even rough estimations of the improvement achievable by including wireless. For all these reasons, we defer the evaluation of potential speedups to later in the project.

3.4 Federated Learning

As in standard DNNs, there are in the order of million parameters that define the model, a large amount of data is required to adjust the parameters of these DNNs. In an environment with distributed computation capabilities, Federated Learning (FL) technique can enable several processors to take in charge part of the computation locally to avoid having to send back sensible data to the cloud and can enable speedup with limited accuracy degradation [18]. Figure 11 presents the general structure of a FL architecture: several clients perform the training process on a reduced dataset and send their updated weights values (W_i) to a server. Then the server performs an averaging operation and stream back the averaged weights values (W_{i+1}) to all the clients.

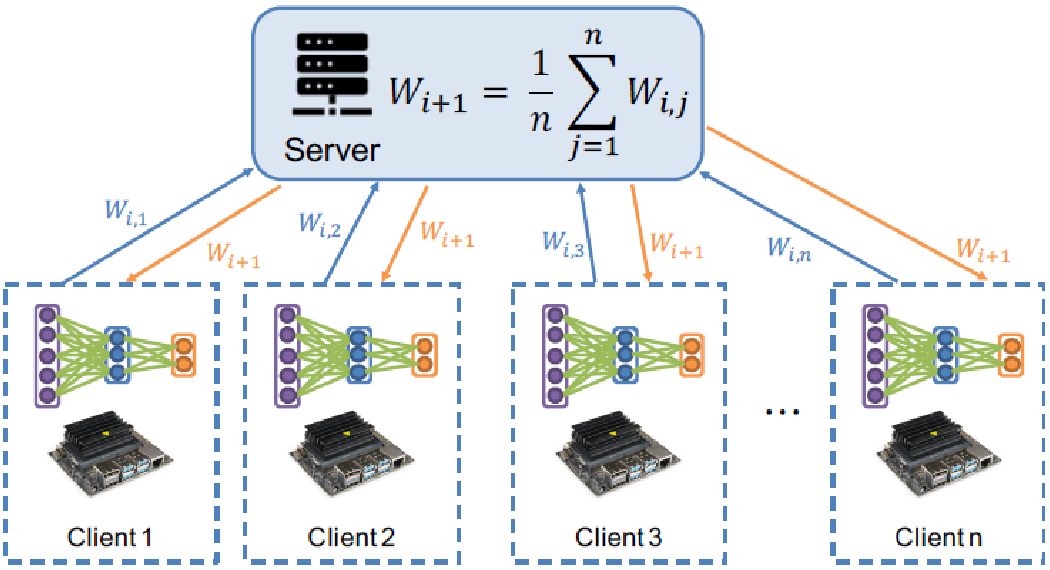

Figure 11. Structure of a Federated Learning architecture with n clients and one server.

While FL is usually considered as a way to maintain privacy in the context of medical applications, in this project it can be considered in various ways:

- In an Edge-level environment (Architecture #1 or Architecture #2), the in-memory computing accelerator and PULP cluster capabilities are leveraged to accelerate the learning process and improve the energy efficiency of the clients. In this case, the bandwidth and improvements are similar to those discussed in previous sections.
- In an HPC environment (Architecture #3), a wireless-enabled MPSoC can contain both the clients and the server in order to: (i) break down the bandwidth limitations by reducing the dataset required by the clients, (ii) leverage the reconfigurable point-to-point communication capability of the wireless interconnect as which of the clients access different data, and (ii) leverage steaming capabilities of wireless interconnect to perform the weight averaging operation in a highly efficient manner. Such operations are very memory demanding, given the huge number of parameters required by state of the art CNNs (e.g.: 4.2M for MobilnetV1). Hence scaling this application to a high number of clients is highly challenging from a communication bandwidth perspective. Considering again MobilnetV1 (with 8-bit weight quantization) and a 64-clients MPSoC, 336MB/s are required to execute a weight update operation in 100 ms.

3.5 Graph Neural Networks

Even though these are important areas which have made them the focus of ANNs, most of the crucial data that is being generated does not come with a regular structure. Hence, in order to exploit the potential of Neural Networks in non-Euclidean data structures such as graphs, Graph Neural Networks (GNN) were introduced in [22]. It is interesting to note that GNNs were initially introduced as a generalization of the RNNs. Further, some of the well-known examples for scenarios modeled with graphs are traffic roads, computer networks, social media interactions, citation networks and chemical molecules.

In a GNN, several neural network algorithms work over the graph structure, along its edges and nodes, with the aim of extracting information in an edge-centered, node-centered or graph-centered manner. That is, depending on the application, one can choose to extract information from the nodes (e.g. whether a node belongs to a shortest-path solution [23]), edges (e.g. delay a link will suffer in a computer network [24]), or the complete graph (e.g. global properties of a molecule [25]). To do so, several types of GNN variants have been proposed: Non-local neural networks (NLNN) capture long-range dependencies of graphs by considering a weighted sum of all nodes (in space and time) taking into consideration the most relevant items by using a pairwise function. Related to them, Graph Attention Networks (GAT) leverage the attention mechanism to concentrate the focus on specific subsets of the graph, which make them worthy in variable input size applications. Relation Networks and Deep Sets are useful to extract global features in graph-centric implementations by focusing only on edge information and node information respectively. However, Message Passing Neural Networks (MPNN), considered a generalization of several previous works [25], are by far the most popular. An MPNN can be used to extract both node and edge embeddings as well as graph embeddings, by applying a message passing phase and a readout phase afterwards to the input graph. A particular class of MPNN are the Graph Convolutional Networks (GCN). Notably though, the same GNN operating principle, which consists of the propagation of a representation of nodes' information along the edges until some defined convergence is achieved, is usually shared. A generalization of the Graph Networks concept, as well as an exhaustive literature review related to it can be found in [23].

Additionally, to highlight the significance of the impact that GNNs have had since their conception, some of the most salient applications that we can find where GNNs have been utilized are the recommender systems, network optimization, community detection, pattern discovery, knowledge graphs, link prediction, properties prediction of structures, etc. Furthermore, some of the most popular datasets that are being utilized by the GNN community, and which will also be the candidates for our study, are Cora, PubMed, Reddit, CiteSeer, PPI, NELL, MNIST, CIFAR-10 and ImageNet.

Given the significant popularity of MPNNs, which are a class of GNNs, in this section we elaborate upon their operational characteristics. Specifically, the inference run is composed by a T-step message passing stage and a readout stage. In the former, two neural networks (message function and update function) work together to allow the graph to dynamically interconnect and update its node and edge features. These NNs are trained so that they extract relevant representations of the neighbor nodes or edges, such as relative situation in the graph or implicit learnings that depend on the application. Therefore, in each of the T time steps, two NN computations are needed for each node. However, we will have extremely large weight sharing opportunities since both NN will be the same in each node (weight tying). Note that, the magnitude of T is a hyper-parameter that can be tweaked until obtaining a reasonable behaviour, although it is highly linked with the Banach Fixed Point (BFP) theorem. Next, in the readout phase, a third neural network gets as input the whole graph obtained in the previous stage and outputs a low-dimension representation of it.

With this background, we now move onto exploring the problem of mapping a GNN in a hardware platform. Multiple challenges exist, as compared to the mapping of classical CNNs, for the case of GNNs.

Firstly, a CNN, which can be understood as a particular GNN example over a regularly structured graph, enables a homogeneous dataflow to be deployed over the architecture. This helps to design a fairly regular architecture, for instance via systolic arrays. However, since GNNs work over irregular data, flexibility can be an asset to reach CNN-like architecture performances. We foresee that having the mechanisms provided by wireless interconnects, to heterogeneously and dynamically distribute workloads over the processing elements, will allow to customize its inference while still exploiting a general architecture.

Further, a GNN will be running typical neural networks such as CNN and RNN models, scheduled in a specific manner to run iteratively over the features of the graph items. To accomplish this task, a control plane will be needed to orchestrate the queued tasks over the available PEs in order to obtain reasonable efficiencies by avoiding unnecessary data movement. For instance, high affinity nodes in the graph may need to exchange large amounts of data whereas lower affinity nodes may not, but this may change dynamically over time. Here, by affinity nodes we mean a set of nodes that are connected by multiple edges and share a strong relation. And so, for such scenarios, a low-latency control plane alongside wireless interconnects can provide the necessary flexibility. For instance, the aforesaid setup can broadcast setup codes from a look-up table. This is similar to the flexible dataflow strategy followed in DNN accelerators like MAERI [35], where the dataflow choice can be change in real-time, but with the advantage of working in GNNs. A particular example of customized location of heterogeneous computation in a GNN accelerator could be by means of different precision/performance PEs. Subsequently, regions of the graph that require higher computational resources could be sent to that part of the accelerator -where more powerful cores are available- depending on the stage of the inference. This would allow for a flexible performance-cost optimization-based strategy.

Additionally, a relevant strategy to follow when mapping GNNs into hardware is to consider graph partitioning, inherited from classical graph theory. In fact, the extremely limited number of works focusing on GNN execution on hardware are highlighting the importance of graph tiling [36, 37]. However, the process of partitioning leads to varying computational domains depending on the input graph and the result of the graph tiling. Moreover, as a consequence, at some point different -- and potentially distant -- processing elements focusing on different tiles may be subject to share their partial outputs due to the high degree of data reusability of these strategies. Specific accelerators that solve these heterogeneous workload mappings are challenging to design, and hence efficient wireless interconnects can potentially provide a convenient solution.

In addition, as we previously stated, large amount of weight reusability is possible in MPNN due to the nature of message and update functions. We foresee that these parameters will be required in a large portion of the architecture cores after being computed and stored in memory. Therefore, the distribution stage could be significantly enhanced by leveraging wireless-enabled broadcast scheme.

Thus, targeting the high performance computing architecture, multiple chips will be available in order to obtain a high computational throughput. Furthermore, in a wireless architecture, the broadcast transmissions, which will be essential for GNNs, are naturally provided by the transceivers without any extra cost as compared to the unicast transmissions. All of these represent intuitive and yet reasonable opportunities WiPLASH has, with regards to tackling the problem of GNN acceleration [36].

Given the relevance of operational characteristics and the target hardware architectures, it is imperative to understand MPNN algorithm and its mathematical structure, so as to be able to determine specific areas for optimization via wireless networks. And so, the MPNN algorithm working over a graph G = (V,E), as defined in [38], is as follows:

```
1: for $v \in V$ do
2:     $\mathbf{h}_v^0 \leftarrow [\mathbf{x}_v, 0, \ldots, 0]$
3: for $t = 1$ to $T$ do
4:     for $v \in V$ do
5:         $\tilde{\mathbf{m}}_v^{t+1} \leftarrow M_t\left(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}\right)$
6:     for $v \in V$ do
7:         $\mathbf{m}_v^{t+1} \leftarrow \sum_{w:(v,w)\in E} \tilde{\mathbf{m}}_v^{t+1}$
8:         $\mathbf{h}_v^{t+1} \leftarrow U_t\left(\mathbf{h}_v^t, \mathbf{m}_v^{t+1}\right)$
9: $\hat{\mathbf{y}} \leftarrow R(\mathbf{h})$
```

where the message function is:

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}) \qquad (1)$$

in which h^t_v denotes the feature vector of node v at time step t, e_{vw} denotes the feature of the edge linking nodes v and w, and N(v) represent the neighbors of node v. M_t is the Message function.

The update function is defined as:

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \qquad (2)$$

where U_t is the Update function. A particular example [38] is:

$$M_t\left(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{wv}\right) = A\left(\mathbf{e}_{wv}\right) \times \mathbf{h}_w^t + b\left(\mathbf{e}_{wv}\right)$$

$$U_t(\mathbf{h}_v^t, \mathbf{m}_v^{t+1}) = GRU(\mathbf{h}_v^t, \mathbf{m}_v^{t+1})$$

where ***A*** and ***b*** are matrix and bias valued ANN, which do not depend on t. GRU represents Gated Recurrent Unit. Therefore, their parameters can be broadcasted to all PEs.

Depending on the message and update functions, the equations above can be rewritten in a matrix form. For instance, the computing of a layer in a GCN with linear message function can be expressed as $X^{(t+1)} = \sigma(A \cdot X^{(t)} \cdot W^{(t)})$, where A is the adjacency matrix, $X^{(t)}$ is the matrix containing feature vectors of all vertices at layer or time step t, and $W^{(t)}$ is the weight matrix at later or time step t. The dimension of A depends on the input graph, whereas the value of X and W, as well as the number of layers, depend on the GNN architecture from input to hidden layers to output.

Model	Graph	#Vertices	#Edges	#Feature/ #Relation	Label
	Cora (CA)	2708	10556	1433	7
GCN	PubMed (PB)	19717	88651	500	3
	Nell (NE)	65755	251550	5415	210
	CoraFull (CF)	19793	126842	8710	67
GS-Pool	Reddit (RD)	232965	114.6M	602	41
	Enwiki (EN)	3.6M	276.0M	300	12
	Amazon (AN)	8.6M	231.6M	96	22
Gated-GCN	Synthetic A (SA)	4.19M	67.1M	100	16
	Synthetic B (SB)	8.38M	134.2M	100	16
GRN	Synthetic C (SC)	12.41M	205.3M	64	16
	Synthetic D (SD)	16.76M	268.4M	50	16
	AIFB (AF)	8285	29043	91	4
R-GCN	MUTAG (MG)	23644	192098	47	2
	BGS (BG)	333845	2166243	207	2
	AM (AM)	1666764	13643406	267	11

Figure 12. Representative set of graphs for different GNNs, from [36]

To illustrate the memory bandwidth requirements, let us take the Cora dataset from Figure 12. Assuming a single layer and 8-bit precisions in all variables, the total data to transfer the three matrices from main memory would be (2708 x 2708) for A, (2708 x 1433) for X, and (1433 x 7) for W. The sum equals to 10.7 MB. Larger matrices would lead to larger transfers, whereas it is also worth noting that some techniques such as graph sampling reduce the burden of loading the adjacency matrix.

In the Cora example, if we want an inference latency on the order of 10 ms, then the required bandwidth is on the order of 1 GB/s. Assuming that the inference latency stays fixed and that we do not use sampling, the requirements at larger graphs can easily increase by orders of magnitude. Not surprisingly, works on GNN acceleration consider HBM modules with bandwidths of 68 GB/s [41], 256 GB/s [36, 39], or 459 GB/s [40].

With the bandwidth figures shown above, it is unlikely that the wireless interconnect will just blindly replace or complement wired networks to increase bandwidth. Instead, with the background about the target architecture and structure of the GNNs now established, we explore the various opportunities that exist in adapting to the new GNN computational requirements as well as in optimizing its inference. Concretely, we present two scenarios where the wireless infrastructure in an HPC can be utilized for the purpose of GNN acceleration.

(i) Heterogeneous scheduling

At different time steps, different regions of our PE array could be working on different regions of the graph, because of the iterative nature of the message passing algorithm or because of different partitioning strategies. Alternatively, some of the PEs could work on solving the message functions, while others can be utilized for the update functions. In all of these cases, flexible scheduling is mandatory, and wireless interconnects can enable either the control plane for it or the data management directly, if bandwidth permits. Figure 13 shows a hypothetical scenario indicative of how the aforementioned strategy can be realized. Concretely, at step “t”, we have 4 partitions of our original graph that are being processed on 4 clusters of our HPC. However, as the algorithm progresses, at step "t+1" the graph has only 3 partitions, wherein the rightmost partition necessitates more resources than the other partitions. In such a scenario, WiPLASH can potentially restructure the original scheduling and assign more processor clusters to the task corresponding to processing the rightmost graph partition.

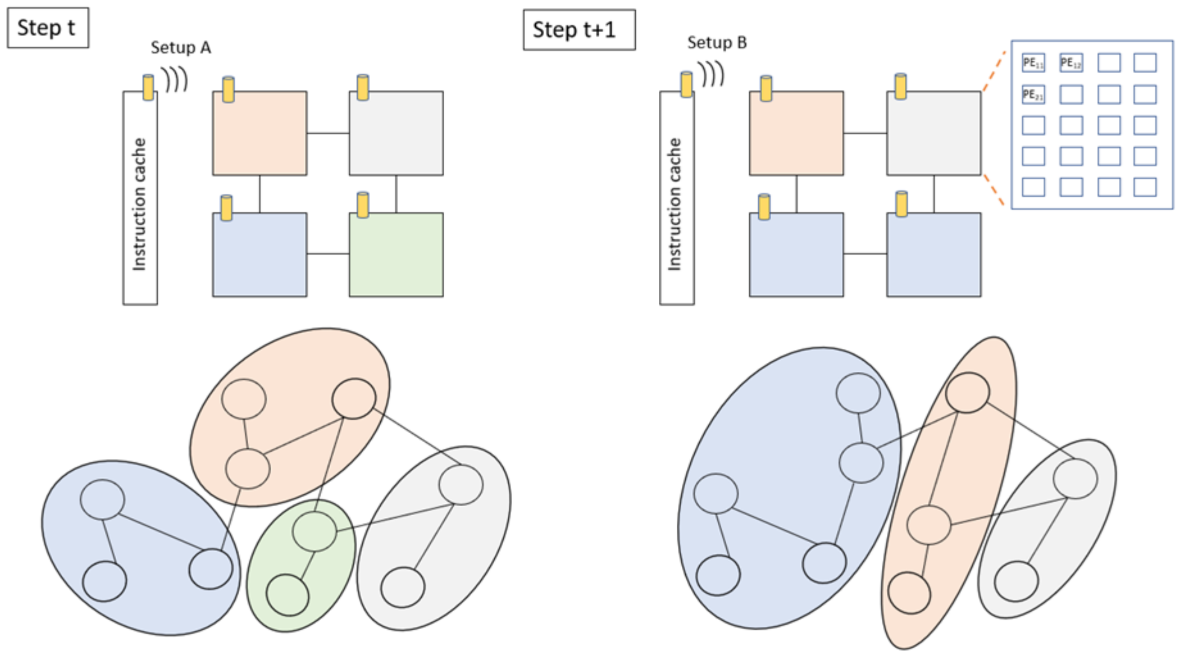

Figure 13. Heterogeneous scheduling managed via wireless interconnects.

(ii) Weight broadcasting

We can leverage wireless broadcast capabilities to distribute message and update NN learnt parameters in a few cycles to all PEs. InFigure 14, different portions of our PE array are working on different nodes and edges, but the weights to be applied in each of them can be broadcasted.

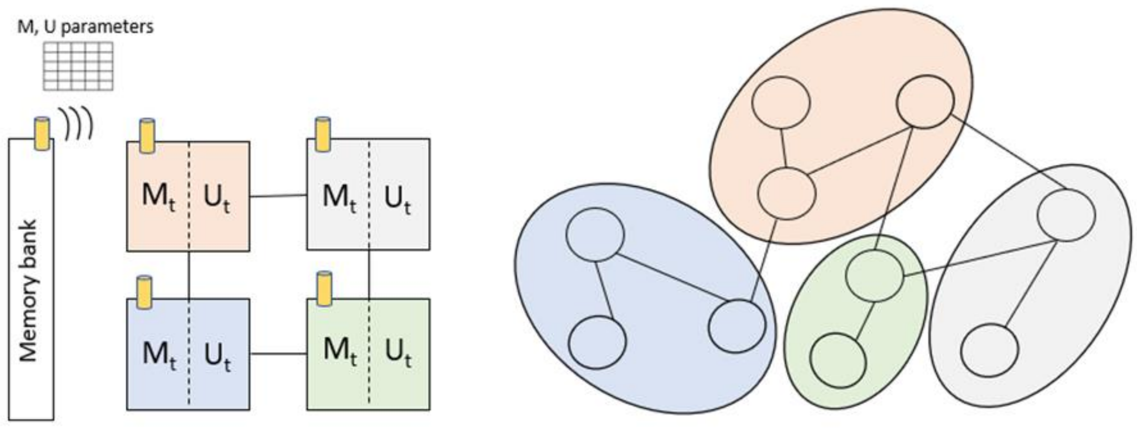

Figure 14. Weight broadcasting using wireless interconnects.

To summarize, for designing a complete GNN hardware platform, we can split the computational work in two layers: an upper layer that orchestrates (partitions and schedules) the in-graph workloads (graph-centric) and a bottom layer that runs and splits the specific neural networks (NN-centric). Considering this approach, a baseline can be obtained by taking estimates from classical DNN accelerators and determining the extra consumption of the graph-level tasks. And so, reference [36], to the best of our knowledge is the only work in literature that tackles directly the problem of GNN acceleration on specific hardware architectures. Hence, given it is the state-of-the-art strategy, we will utilize it as our baseline for comparison.

Evaluating the acceleration in a GNN is complex even in specialized hardware systems because the optimal dataflow depends on the characteristics of the graph and of the GNN layers. This, together with the non-trivial role of the wireless interconnect in adapting the accelerator/HPC platform to the most optimal dataflow, complicates the estimation of gains related to wireless technology. Therefore, we defer this task to preliminary simulations later in the project.

3.6 Next Generation Sequencing

Genome sequencing is also one of the first steps in understanding a new disease, its effects, development of diagnostic tests and possible cure and vaccines for it. This process should be quite quick and efficient is case of a new disease outbreak, in order to curtain it, as has been evident during the recent outbreak of the novel coronavirus (COVID-19) [42].

Genome sequencing is the process of determining the DNA sequence or the order of bases As, Cs, Gs, and Ts making up the organism's genome. Next-generation sequencing (NGS) is a high-throughput genome sequencing method. In sequence alignment, which is one of the first steps in NGS, a sequence read is aligned or checked against a genomic reference for regions of similarity [43]. This process must tolerate differences between the query read and the reference genome, due to errors in the sequencing process and genuine differences between organisms.

There are several optimized sequence alignment NGS applications, including some of the most widely used, relying upon the FM-index data structures and search algorithms. Bowtie2 [44], BWA-MEM [45] and HISAT2 [46] are three of the most important state-of-the-art widely used sequencing application that use FM-Index.

The FM-index is a data structure that allows fast substring searches over large texts [47]. FM-index is based on several data structures and algorithms, such as Suffix Array and Burrows-Wheeler Transform (BWT). A diagram of the flow of an FM-index search is depicted in Figure 15.

The most suitable architecture for this application seems to be Architecture #3, as this application has HPC requirements. Furthermore, this application performs a very large numbers of random memory reads for performing the searches, so it requires a very high memory bandwidth (to main memory, most likely, as the genome is quite large), together with low latency. Indeed, our explorations of genomic sequencing algorithms [48] highlighted that peak memory bandwidths of 17GB/s 10GB/s and 8 GB/s were required by Bowtie2, BWA-MEM and HISAT2 (respectively) to outperform the state of the art Intel KNL system when employing clusters of ARM processors. For these reasons, improvement may not come by the sheer increase of bandwidth, but also from architectures capable of coping with the irregular accesses to main memory. For this reason, it is not trivial to infer the exact role of the wireless interconnect and its impact on the performance and efficiency of genome sequencing in wireless-enabled HPC architectures. For this, we will defer the assessment of potential to later in the project.

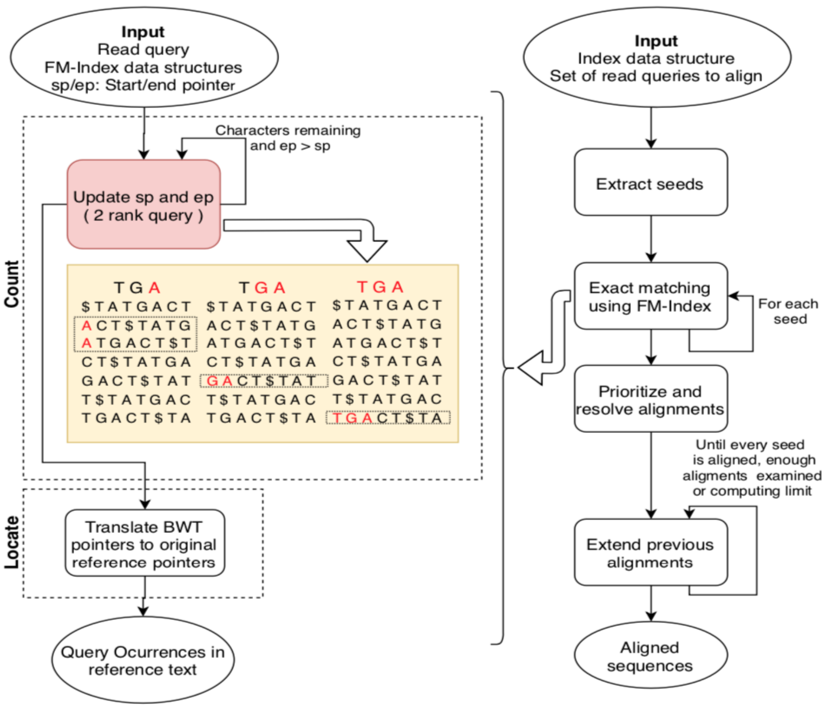
Input
Read query
FM-Index data structures
sp/ep: Start/end pointer
Characters remaining and ep > sp
Update sp and ep (2 rank query)
Count
Locate
Translate BWT pointers to original reference pointers
Query Ocurrences in reference text
Input
Index data structure
Set of read queries to align
Extract seeds
Exact matching using FM-Index
For each seed
Prioritize and resolve alignments
Until every seed is aligned, enough aligments examined or computing limit
Extend previous alignments
Aligned sequences

Figure 15. FM-index search

3.7 Real-Time Video Surveillance

The real-time video surveillance application proposes the use of an environment consisting on the deployment of Ultra High definition cameras (edge devices) that record video in 4K resolution combined with a H265 encoding that reduces the frame size to almost the half than H264. The cameras, besides being the recording system, are also part of the computing system that will perform image recognition inference combining the computation capabilities of the edge devices (cameras in this case) with edge data centers (DCs), when needed. Video transcoding and streaming represents today 58% of the overall downstream traffic [49], whereas real-time video analytics (powered by CNNs) are becoming one of the main “killer applications” for artificial intelligence, with already a great impact in various domains of our daily life [50].

Real-time H265 video encoding is performed by means of the Kvazaar [51] real-time video transcoder application, whereas image classification is performed with state-of-the-art CNNs (such as Alexnet, MobileNet v1 and v2, and ResNet50) due to their current relevance in edge devices.

Both encoding and classification are memory-intensive. Moreover, they have to abide to real-time constraints. These characteristics put stringent requirements on the interconnect, as typically video streams can be encoded at high frame rates (up to 24FPS) and resolutions (up to 640x480 pixels) for high-performance video analytics scenarios such as drone navigation and autonomous driving applications. Indeed, for the aforementioned settings, we measured a peak bandwidth of 16 GB/s.

4 Concluding Remarks

This deliverable identified applications workloads to be used in the project as case studies. First, it introduced the hardware architectures that will be evaluated in the project, belonging to three different domains: specialized in-memory computing-based hardware accelerators, massively parallel multi-core scalable platforms, and more traditional HPC systems. Second, it introduced and presented the target applications for the project, together with key performance indicators related to the implementation and optimization of the proposed applications to the target platforms. The quantitative evaluations related to application mapping and optimization on the target WiPLASH platforms will be provided in D4.3, but here we make some preliminary remarks.

We have chosen a set of benchmarks with the focus on DNNs, but showing a wide range of characteristics that make them suitable for different architectures. Starting with DNNs for embedded systems and specialized accelerators, to genome sequencing or recommendation systems more suitable to HPC systems. The bandwidth requirements vary widely, from a few tens of Mb/s in embedded DNN inference to Tb/s ranges in DLRM or GNNs.

Taking into consideration that conservative assumptions lead to considering wireless interconnects with a few tens of Gb/s bandwidth, and more aggressive ones pointing to a few hundreds of Gb/s, then it is clear that the wireless interconnect will not simply replace or augment the wired counterpart for reasons of bandwidth. Instead, the low latency, system-level flexibility and inherent broadcast capabilities should be put in action to enhance the architecture. In fact, the broadcast/multicast capability alone can represent a boost of the effective impact of wireless as a single broadcast channel at 1 Gb/s just requires a transmission of 1 Gb/s regardless of the number of receivers. In the wired case, on the contrary, addressing N cores at 1 Gb/s requires a much larger bisection bandwidth because it requires the delivery of N packets. Further, in-memory computing arrays can exploit the reconfigurability to implement adaptive accelerators able to optimally map multiple CNNs without requiring overprovisioning at the interconnect.

The potential gains in efficiency and performance of introducing wireless interconnects depend on the application and architecture. On the one hand, the evaluation is straightforward in the embedded DNN and the estimated gains can easily reach 10X in efficiency (inference and training) and performance (in training only, because inference does not require very high bandwidth). On the other hand, the architectural considerations make it relatively difficult to evaluate the potential gains of including a wireless interconnect in the rest of architectures. However, our previous experience in wireless-enabled architectures [52] tells us that, depending on the application characteristics and the architecture, improvements can reach 10X as well. In any case, we defer the evaluation of HPC architectures to later in the project, when the simulator will be available for architectural explorations, and rather focus the work on the DNN acceleration initially. Later in the project, the other applications shall be explored through batch simulations in a few selected HPC architectures that do not require fine-tuning or mapping of the application on specialized hardware.

Bibliography

[1] A. Pullini, D. Rossi, I. Loi, G. Tagliavini and L. Benini, "Mr. Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing," in *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, pp. 1970-1981, July 2019.

[2] RISC-V ISA Specifications, https://riscv.org/specifications

[3] M. Gautschi et al., "Near-Threshold RISC-V Core with DSP Extensions for Scalable IoT Endpoint Devices," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 10, pp. 2700-2713, Oct. 2017.

[4] F. Conti, P. D. Schiavone and L. Benini, "XNOR Neural Engine: A Hardware Accelerator IP for 21.6-fJ/op Binary Neural Network Inference," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 11, pp. 2940-2951, Nov. 2018.

[5] Y. M. Qureshi, W. A. Simon, M. Zapater, D. Atienza and K. Olcoz, "Gem5-X: A Gem5-Based System Level Simulation Framework to Optimize Many-Core Platforms," 2019 Spring Simulation Conference (SpringSim), Tucson, AZ, USA, 2019, pp. 1-12.

[6] A Sridhar, A Vincenzi, D Atienza, T Brunschwiler, 3D-ICE: a compact thermal model for early-stage design of liquid-cooled ICs, IEEE Transactions on Computers, Vol 63, No. 10, October 2014.

[7] D. Rossi et al., "Application Space Exploration of a Heterogeneous Run-Time Configurable Digital Signal Processor," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, no. 2, pp. 193-205, Feb. 2013.

[8] Cypress Semiconductors HyperRAM:
https://www.cypress.com/products/hyperram-memory

[9] APMEM IoT RAM Products: http://www.apmemory.com/html/product_psram.php

[10] GAP-8: IoT processor:
https://greenwaves-technologies.com/ai_processor_gap8/product-brief-form/

[11] V. Sze, Y. Chen, T. Yang, and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” Proc. IEEE, vol. 105, no. 12, pp. 2295–2329, 2017.

[12] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016, [Online]. Available: http://www.deeplearningbook.org

[13] Li Fei-Fei, R. Fergus and P. Perona, "One-shot learning of object categories," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 4, pp. 594-611, April 2006.

[14] J. Snell, K. Swersky, and R.S. Zemel, “Prototypical Networks for Few-shot Learning”, ArXiv170305175 Cs Stat, 2017.

[15] T. Lesort, et al. “Continual Learning for Robotics”. ArXiv190700182 Cs, 2019.

[16] D. Maltoni, and V. Lomonaco, “Continuous learning in single-incremental-task scenarios” Neural Netw, Vol. 116, no. 56–73, 2019.

[17] U. Gupta, X. Wang, M. Naumov, C.-J. Wu, B. Reagen, D. Brooks, B. Cottel et al. "The architectural implications of Facebook's DNN-based personalized recommendation." arXiv preprint arXiv:1906.03109, 2019.

[18] K. Chen and Q. Huo, "Scalable Training of Deep Learning Machines by Incremental Block Training with Intra-block Parallel Optimization and Blockwise Model-Update Filtering," IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2016.

[19] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), Kuala Lumpur, 2015, pp. 730-734.

[20] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object detection via region-based fully convolutional networks," in Proceedings of the NIPS, 2016.

[21] A. Toshev et al., "Deeppose: Human pose estimation via deep neural networks," in CVPR, 2014.

[22] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner and G. Monfardini, "The Graph Neural Network Model," in IEEE Transactions on Neural Networks, vol. 20, no. 1, pp. 61-80, Jan. 2009.

[23] P. W. Battaglia et al., "Relational inductive biases, deep learning, and graph networks," 2018.

[24] K. Rusek, et al. "Unveiling the Potential of Graph Neural Networks for Network Modeling and Optimization in SDN." Proceedings of the 2019 ACM Symposium on SDN Research, 2019.

[25] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural Message Passing for Quantum Chemistry," in Proceedings of the ICML '17, 2017, pp. 1263–1272.

[26] A. K. McCallum, "Automating the construction of internet portals with machine learning." Information Retrieval. 3. 127-163, 2000.

[27] C. Giles, K. Bollacker, and S. Lawrence, Steve, "CiteSeer: An Automatic Citation Indexing System," Proceedings of 3rd ACM Conference on Digital Libraries, 2000. 10.1145/276675.276685.

[28] H. Dai, Z. Kozareva, B. Dai, A. Smola, L. Song; "Learning Steady-States of Iterative Algorithms over Graphs"; Proceedings of the 35th International Conference on Machine Learning, PMLR 80:1106-1114, 2018.

[29] J. Baumgartner, S. Zannettou, B. Keegan, M. Squire, J. Blackburn, The Pushshift Reddit Dataset. Zenodo. 2020. http://doi.org/10.5281/zenodo.3608135

[30] M. Zitnik and J. Leskovec, "Predicting multicellular function through multi-layer tissue networks," Bioinformatics, vol. 33, no. 14, pp. i190– i198, 2017.

[31] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, and T. M. Mitchell. 2010. Toward an architecture for never-ending language learning. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI'10). AAAI Press, 1306–1313.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, November 1998.

[33] A. Krizhevsky. "Learning multiple layers of features from tiny images." Tech Report, 2009.

[34] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge." International Journal of Computer Vision, 2015.

[35] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects," in Proceedings of the ASPLOS '18, 2018, pp. 461–475.

[36] S. Liang, et al. "EnGN: A High-Throughput and Energy-Efficient Accelerator for Large Graph Neural Networks", IEEE Transactions on Computers, 2020.

[37] L. Ma et al., “Neugraph: Parallel deep neural network computation on large graphs,” Proc. 2019 USENIX Annu. Tech. Conf. USENIX ATC 2019, pp. 443–457, 2019.

[38] K. Rusek and P. Chołda, “Message-Passing Neural Networks Learn Little’s Law,” IEEE Commun. Lett., vol. 23, no. 2, pp. 274–277, 2018.

[39] M. Yan *et al.*, “HyGCN: A GCN Accelerator with Hybrid Architecture,” in *Proceedings of the HPCA ’20*, 2020.

[40] T. Geng *et al.*, “AWB-GCN: Accelerating Graph Convolutional Networks through Runtime Workload Rebalancing,” in *Proceedings of MICRO-53*, 2020.

[41] A. Auten, M. Tomei, and R. Kumar, “Hardware Acceleration of Graph Neural Networks,” in *Proceedings of the DAC ’20*, 2020.

[42] “Whole genome of the Wuhan coronavirus, 2019-nCoV, sequenced,” 2020. [Online]. Available: www.sciencedaily.com/releases/2020/01/200131114748.htm

[43] H. Li and N. Homer, “A survey of sequence alignment algorithms for next-generation sequencing,” Brief Bioinform., pp. 473–483, May 2010.

[44] B. Langmead and S. L. Salzberg, “Fast gapped-read alignment with bowtie 2,” Nature Methods, pp. 357–359, Mar 2012.

[45] H. Li and R. Durbin, “Fast and accurate short read alignment with burrows-wheeler transform,” Bioinformatics, vol. 25, no. 14, pp. 1754–1760, May 2009.

[46] D.Kim, J.M.Paggi, C.Park, C.Bennett, and S.L. Salzberg,“Graph-based genome alignment and genotyping with HISAT2 and HISAT genotype,” Nature Biotechnology, vol. 37, no. 8, pp. 907–915, Aug. 2019.

[47] P. Ferragina and G. Manzini, “Opportunistic data structures with applications,” in Proceedings 41st Annual Symposium on Foundations of Computer Science, 2000, pp. 390–398.

[48] Y. M. Qureshi et al., "Genome Sequence Alignment - Design Space Exploration for Optimal Performance and Energy Architectures," in IEEE Transactions on Computers, 2020.

[49] Sandvine 2018. “Global Internet Phenomena Report. 2018”. URL: https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf

[50] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha. 2017. “Real-Time Video Analytics: The Killer App for Edge Computing”. Computer, pp. 58–67.

[51] M. Viitanen, A. Koivula, A. Lemmetti, A. Ylä-Outinen, J. Vanne, and T. D. Hämäläinen. 2016. “Kvazaar: Open-Source HEVC/H. 265 Encoder”. In Multimedia Conference, pp. 1179–1182.

[52] V. Fernando, A. Franques, S. Abadal, S. Misailovic, J. Torrellas, “Replica: A Wireless Manycore for Communication-Intensive and Approximate Data,” in Proceedings of the ASPLOS ’19, Providence, RI, USA, April 2019.