Manuscript Details

Manuscript number	MICPRO_2019_21
Title	A Technologically Agnostic Framework for Cyber-Physical and IoT Processing-in-Memory-based Systems Simulation
Article type	Research Paper

Abstract

Smart devices based on Internet of Things (IoT) and Cyber-Physical System (CPS) are emerging as an important and complex set of applications in the modern world. These systems can generate a massive amounts of data, due to the enormous quantity of sensors being used in modern applications, which can either stress the communication mechanisms or need extra resources to treat data locally. In the era of efficient smart devices, the idea of transmitting huge amounts of data is prohibitive. Furthermore, implementing traditional architectures imposes limits on achieving the required efficiency. Within the area, power, and energy constraints, Processing-in-Memory (PIM) has emerged as a solution for efficiently processing big data. By using PIM, the generated data can be processed locally, with reduced power and energy costs, allowing an efficient solution for CPS and IoT data management problem. However, two main tools are fundamental on this scenario: a simulator that allows architectural performance and behavior analysis is essential within a project life-cycle, and a compiler able to automatically generate code for the targeted architecture with obvious improvements of productivity. Also, with the emergence of new technologies, the ability to simulate PIM coupled to the latest memory technologies is also important. This work presents a framework able to simulate and automatically generate code for IoT PIM-based systems. Also, supported by the presented framework, this work proposes an architecture that shows an efficient IoT PIM system able to compute a real image recognition application. The proposed architecture is able to process 6x more frames per second than the baseline, while improving the energy efficiency by 30x.

Keywords	IoT; CPS; Processing-in-Memory; Simulation; Compiler
Taxonomy	Emergent Computing, Internet of Things, Hardware Architecture, Cyber-Physical System

Corresponding Author	Paulo Cesar Santos
Corresponding Author's Institution	UFRGS
Order of Authors	Paulo Cesar Santos, João Paulo C. de Lima, Rafael F. de Moura, Hameeza Ahmed, Marco Antonio Zanata Alves, Antonio Carlos Schneider Beck Filho, Luigi Carro

Submission Files Included in this PDF

File Name [File Type]

SI_INTESA_submitted_v2.pdf [Manuscript File]

PC.jpg [Author Photo]

joao.jpg [Author Photo]

Hameeza.pdf [Author Photo]

rafael.pdf [Author Photo]

marco.jpg [Author Photo]

luigi.jpg [Author Photo]

Submission Files Not Included in this PDF

File Name [File Type]

pc.txt [Author Biography]

joao.txt [Author Biography]

rafael.txt [Author Biography]

Hameeza.txt [Author Biography]

marco.txt [Author Biography]

antonio.txt [Author Biography]

luigi.txt [Author Biography]

antonio.png [Author Photo]

To view all the submission files, including those not included in the PDF, click on the manuscript title on your EVISE Homepage, then click 'Download zip file'.

A Technologically Agnostic Framework for Cyber-Physical and IoT Processing-in-Memory-based Systems Simulation

PAULO CESAR SANTOS*, Institute of Informatics, UFRGS, Brazil
JOÃO PAULO C. DE LIMA, Institute of Informatics, UFRGS, Brazil
RAFAEL F. DE MOURA, Institute of Informatics, UFRGS, Brazil
HAMEEZA AHMED, Dept. Comp. and Inf. Eng, NED, Pakistan
MARCO A. Z. ALVES, Institute of Informatics, UFPR, Brazil
ANTONIO C. S. BECK, Institute of Informatics, UFRGS, Brazil
LUIGI CARRO, Institute of Informatics, UFRGS, Brazil

Abstract - Smart devices based on Internet of Things (IoT) and Cyber-Physical System (CPS) are emerging as an important and complex set of applications in the modern world. These systems can generate a massive amounts of data, due to the enormous quantity of sensors being used in modern applications, which can either stress the communication mechanisms or need extra resources to treat data locally. In the era of efficient smart devices, the idea of transmitting huge amounts of data is prohibitive. Furthermore, implementing traditional architectures imposes limits on achieving the required efficiency. Within the area, power, and energy constraints, Processing-in-Memory (PIM) has emerged as a solution for efficiently processing big data. By using PIM, the generated data can be processed locally, with reduced power and energy costs, allowing an efficient solution for CPS and IoT data management problem. However, two main tools are fundamental on this scenario: a simulator that allows architectural performance and behavior analysis is essential within a project life-cycle, and a compiler able to automatically generate code for the targeted architecture with obvious improvements of productivity. Also, with the emergence of new technologies, the ability to simulate PIM coupled to the latest memory technologies is also important. This work presents a framework able to simulate and automatically generate code for IoT PIM-based systems. Also, supported by the presented framework, this work proposes an architecture that shows an efficient IoT PIM system able to compute a real image recognition application. The proposed architecture is able to process 6× more frames per second than the baseline, while improving the energy efficiency by 30×.

Additional Key Words and Phrases: IoT, CPS, Processing-in-Memory, Simulation, Compiler

1 INTRODUCTION

Smart devices have emerged as the new frontier in terms of modern applications. Being widely applied in different environments, Internet of Things (IoT) and Cyber-Physical Systems (CPSs) are present varying from simple systems that monitor temperature, control illumination, or turn on/off a secondary device, to complex applications as medical analyses, driving assistance, image recognition, autonomous vehicles, and drones. Supplied by a large number of sensors, smart devices are currently requiring the management of large amounts of data, and high processing power. Moreover, the embedded nature of these systems requires greater attention to energy efficiency.

*This study was financed in part by CAPES (Finance Code 001), CNPq and FAPERGS

Authors' addresses: Paulo Cesar Santos, Institute of Informatics, UFRGS, Porto Alegre, Brazil, pcssjunior@inf.ufrgs.br; João Paulo C. de Lima, Institute of Informatics, UFRGS, Porto Alegre, Brazil, jpclima@inf.ufrgs.br; Rafael F. de Moura, Institute of Informatics, UFRGS, Porto Alegre, Brazil, rfmoura@inf.ufrgs.br; Hameeza Ahmed, Dept. Comp. and Inf. Eng, NED, Karachi, Pakistan, hameeza@neduet.edu.pk; Marco A. Z. Alves, Institute of Informatics, UFPR, Curitiba, Brazil, mazalves@inf.ufpr.br; Antonio C. S. Beck, Institute of Informatics, UFRGS, Porto Alegre, Brazil, caco@inf.ufrgs.br; Luigi Carro, Institute of Informatics, UFRGS, Porto Alegre, Brazil, carro@inf.ufrgs.br.

Recently, as the cost per transistor has been reduced exponentially over several years [56], more complex embedded systems have emerged to support and improve even more elaborate CPS and IoT applications, which are known by the increasing volume of data and the need to enrich them. To process such information that all these different sensors collect, and to perform the increasingly complex operations present in the modern applications, expensive and sophisticated computation elements, like multi-core General Purpose Processors (GPPs) and large Graphics Processing Units (GPUs) are commonly used. For instance, self-driving vehicles apply these expensive resources for image recognition, pedestrian detection, etc [11]. The same can be noticed for autonomous drones, and modern experimented designs such as LG 16-cameras assay [43]. Moreover, one essential computation required in this system is executing machine learning algorithms, as deep learning and reinforced learning, to make choices based on the environment it is inserted [23, 25, 31, 53, 65].

Several studies have presented different approaches to process such algorithms efficiently, and the most suitable presented in the literature is based on Processing-in-Memory (PIM) designs [9, 22, 33, 49, 61]. PIM architectures have been presented since 1990's [35] targeting the reduction of unnecessary data movement by allocating processing units close to data. With the advent of 3D-stacked memories, and further with the ability of stacking logic and memories on same chip [28, 37], PIM appears as a prominent solution able to keep the compromise between performance and energy. In addition to the ability to reduce data movement, improve energy efficiency and performance, PIM can also take advantage of new memory technologies [12, 60]. Since some proposals such as [49, 55] allow direct access to the new memory technology devices, they take advantage of extracting higher amounts of memory bandwidth, Data-Level Parallelism (DLP) and Floating Point Operations Per Second (FLOPS) from these devices.

However, previous designs lacked appropriate automation tools, since all previous PIM designs have been made based on bare metal with custom strategies. Moreover, two main issues were missing on previous studies:

- on the **architectural** side, how to connect several sensors to be processed in an efficient way by a PIM architecture, and how to allow the PIM to efficiently process all data collected by these sensors;
- on the **design** side, how to provide a design space exploration environment, and how to experiment and evaluate full-stack solutions for CPS and IoT using complete tools to simulate PIM designs and new technologies

This work extends the one in [54], presenting a design framework that comprises a GEM5-based simulator [10] and a LLVM-based compiler [36] as a tool to ease the development of PIM architectural projects for IoT and CPS applications. Also, we show an architectural approach to connect several CPS and IoT devices to a PIM component, being able to share the PIM processing resources, while taking advantage of the known Hybrid Memory Cube (HMC) module. This tool simulates the proposed architecture (host processor, PIM accelerator, and sensors) using optimized binary codes generated by the compiler for the target PIM. We demonstrate the usage of our framework in a case study and the gains of performance and energy obtained using the proposed architecture to run an object detection algorithm.

2 BACKGROUND

In this section, the basics of 3D-stacked memory technology and Processing-in-Memory (PIM) are presented.

2.1 3D-Stacked

3D Integrated Circuits (ICs) and 3D-stacked memories have emerged as a feasible solution to tackle the memory wall problem and the little performance-efficiency improvement achieved by traditional

commodity Dynamic Random Access Memories (DRAMs). By connecting DRAM dies and logic layer on top of each other using dense Through-Silicon Via (TSV), 3D-stacked memories can provide high bandwidth, low latency, and significant energy-efficiency improvements in comparison to traditional Double Data Rate (DDR) modules. The most known examples of 3D-stacking technologies from industry are the Microns's Hybrid Memory Cube (HMC) [28] and AMD/Hynix's High Bandwidth Memory (HBM) [37].

Figure 1 shows an overview of the internal organization of a 3D-stacked DRAM device. For both HMC and HBM architectures, it consists of multiple layers of DRAM, each layer containing various banks. A vertical slice of stacked layers composes a *vault*, which is connected by an independent TSV bus to a *vault controller* [28]. Since each *vault controller* operates its *vault* memory region independently, it enables *vault*-level parallelism similar to independent channel parallelism found in conventional DRAM modules. In addition to the *vault* parallelism, the *vault controller* can share the TSV bus among the layers via careful scheduling of the requests which enables bank-level parallelism within a *vault* [66].

According to the last specification [28], the HMC module contains either four or eight DRAM dies, and one logic layer stacked and connected by a TSV. Each memory cube contains 32 *vaults* and each *vault* controller is functionally independent to operate upon 16 memory banks. The available bandwidth from all *vaults* is up to 320 GBps and it is accessible through multiple serial links. Moreover, the HMC specifies atomic command requests which enable the logic layer to perform read-update-write operations atomically on data using up to 16-byte operands. All in-band communication across a link is *packetized* and there is no specific timing associated with memory requests, since *vaults* may reorder their internal requests to optimize bandwidth and reduce average access latency.

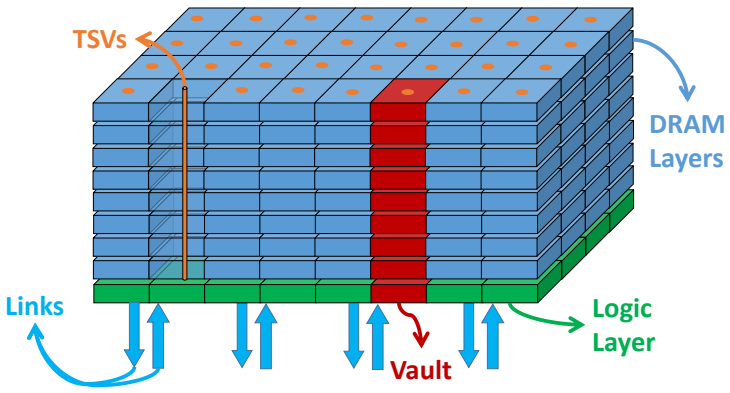

Fig. 1. Layout of a HMC-like device comprising of eight DRAM layers and a base logic layer connected by TSVs and vertically organized in *vaults* [28].

2.2 PIM

The increasing demand for data-intensive applications with ever-growing workloads, and the possibility of having 3D-stacked memory devices leveraged and reintroduced the PIM research field. Data-intensive applications can benefit from PIM since they can compute as near as possible where the data resides, instead of passing the data back and forth through a slow storage device, main memory and cache memories to finally be processed by the processing unit by itself [66].

In the past years, several works have studied how to couple some processing logic in the memory system. Some of them include processing units on the memory controller and DRAM module, which reduces the cost of PIM integration, avoids costly 3D-stacking technology, and uses unmodified DRAM chips. Others propose processing logic within the memory chip or memory array, which significantly improves computational efficiency by taking advantage of the highest bandwidth and the lowest latency possible directly from DRAM banks.

Meanwhile, several works [34, 41] studied the possibility of stacking memory dies and interconnecting them through very small via, granting the emerging of 3D-stacked processing-in-memory approach. Moreover, the evolution of TSV technique [16] solved some problems present on previous versions of 3D-stacked memory like thermal dissipation influences making feasible the production and exploitation of stacked memories as done by the HMC [30, 50] and HBM [37] products. Both HMC and HBM designs separate logic from memory, and deal with the old problem of using the same slow DRAM technology to build logic processing elements.

Consequently, since 2013 3D-stacked PIMs have regained focus with different project approaches, varying from multicore systems placed into the logic layer as presented in [2, 8, 17, 51, 57], alternative cores [32, 47], Single Instruction Multiple Data (SIMD) units [49, 55], Graphics Processing Units (GPUs) [64] to Coarse-Grain Reconfigurable Arrays (CGRAs) [21].

3 RELATED WORK

This section presents state-of-art PIM works regarding their feasibility for big sensor data applications realm, how to compile and simulate for PIM architecture design.

3.1 PIM feasibility for big sensor data applications

Big sensor data applications are built on the premise that data will be collected from different sources (audio, video, biological signals) for analysis and decision-making. Personal assistants, flying drones, and self-driving vehicles are some of those applications that involve a massive amount of data, and they must be processed on the device mainly due to low latency and privacy requirements. However, traditional embedded systems architectures do not fulfill the energy requirements for on-device processing of complex applications, such as machine learning, and it opens up a challenge to bring low-power, energy-efficient, specialized hardware to Cyber-Physical System (CPS) and Internet of Things (IoT) devices.

Considering this class of applications, a significant part of the time is spent processing machine learning algorithms and managing data [4]. Since most of the big sensor data analysis and decision-making is based on statistical and artificial intelligence algorithms, a viable approach to reduce time and energy is by optimizing them to a more data-centric and application-specific architecture. Several previous works have already implemented distinct PIM architectures aiming to both explore the abundant internal memory bandwidth and reduce data movement through the memory system [2, 5, 7, 18, 27, 47]. In particular, previous works as [9, 22, 33, 49, 61, 63] have taken advantage of these new memory architectures to accelerate unsupervised learning and IoT applications in distinct ways.

3.2 Simulating a PIM-based architecture

Most of the recent PIM works focus on fully-programmable cores, which are generally simulated by adjusting constraints of 3D integrated circuits in existing simulators and by taking advantage of existing execution models and compilers. To lower the time spent investigating new PIM techniques for research purposes, the majority of the validation of PIM proposed architectures are based on simulation mechanisms. Additionally, simulating new hardware designs contributes to the reduction and discovering of design

faults which could be detected only after the manufacturing process. Thus, the adoption of simulation environments significantly adds to the PIM design and validation tests. On the other hand, fixed-function in-memory processing, which includes the HMC 2.0 and the works of [3, 20, 46, 48], relies on a more varied design methodology which generally includes custom or in-house tools.

Although there exist numerous PIM simulators, they still lack dealing with many challenges and difficulties in the PIM simulation. The first issue corresponds to the necessity of coupling a significant number of different tools to represent a whole computing system and its respective modules. In [62], the authors presented a PIM simulator that relies on the integration of three memory simulators to support different memory technologies and one architectural simulator to provide interconnection and description of Central Processing Unit (CPU) architectures. Likewise, in [63] is presented a PIM architecture for wireless IoT applications which relies on the integration of one simulator for simulating both PIM and host processing elements and a tool for estimating power consumption. Coupling several simulators to represent the desired computing system incurs drawbacks to the design life-cycle making this simulation approach prohibitive. When considering different simulation environments, the architectural designers must have complete and in-depth knowledge about the simulators features, which in turn demands time-consuming tasks. Additionally, interface and communication protocols must be created and implemented to synchronize all the modules and simulators utilized. Also, since the involved simulators may have different accuracy levels, system modeling patterns, and technological constraint representations, the result of the simulation might not present the desired precision. Although [63] utilizes the same architectural simulator for all the hardware components, different simulation accuracy level components are instantiated to compose the whole system. Thus, the simulation approach followed by [63] not only needs a particular synchronization mechanism but also does not reflect a real scenario where the host processor is represented by an event-detailed processor description and the PIM elements are described only with atomic and no-delayed operations.

Meanwhile, other simulators require the generation of trace files as input to feed them. The major drawbacks inherit from the trace-based simulation approach are the necessity of previous execution of the target applications in a real machine and the gathering of relevant information such as executed instructions and data access addresses. Although [62] and [48] are built over architectural cycle-accurate simulators, the PIM modeling and measurements are done by analyzing memory traces gathered during the simulation.

3.3 Compiling for PIM

Despite the existence of significant work on PIM architectures research, compiler-based solutions for PIM is still a not completely covered subject. Regarding the generation of binaries for PIM architectures, a compiler must deal at least with three main challenges: the offloading of the instructions, the efficient hardware resources exploitation in the memory logic layer, and the programmability. For the offloading decision problem, the compiler must be able to decide when migrating a portion of code and its respective instructions to execute in the PIM logic layer. To maximize the performance and energy improvements obtained by the PIM, the compiler has to exploit the available hardware resources efficiently. Respecting to programmability for PIM, all programmer interventions such as code notation and pragmas or the usage of special libraries are not desired and must be avoided not to disrupt the software development process.

Specifically for offloading decisions, [24] presents an offloading technique for PIM. However, in [24], the offload decisions are taken offline in a non-automatic way due to its necessity of cache profiler and trace analysis. Similarly, [26] introduces a compile-time offloading system candidate for a PIM. Nonetheless, in

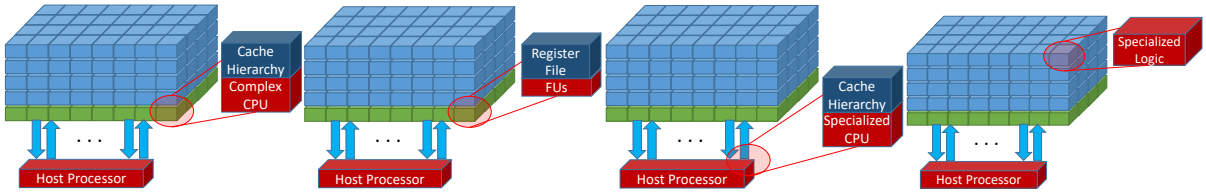

Fig. 2. Types of PIM.

[26] the programmer is required to insert code notations in the portions of code that have potential to be executed in the PIM device hurting the programmability issue. Concerning hardware resources allocation, [2], and [62] propose compiler techniques for PIM architectures, but explicit code notations are requested for mapping the PIM units that will execute the code.

4 A FRAMEWORK FOR PIM SIMULATION

The ability to simulate and automatically generate code for experimental systems is crucial for the development of state-of-the-art devices. In this section we describe the proposed framework that comprises a GEM5-based [10] simulator and an LLVM-based compiler [36]. The presented simulator arrangement is able to simulate different types of PIM architectures, as presented in Section 4.1, and also different types of memory technologies as shown in Section 4.3 by using the HMC organization as basis.

4.1 Simulating a PIM-based system

Since simulating applications on state-of-the-art architectures is an important issue for modern designs, this work takes advantage of the well diffused high-level, event-driving GEM5 simulator [10] to allow the simulation of 3D-stacked-based memories and state-of-the-art PIM logics.

As presented in Section 2, the HMC memory is widely chosen for PIM designs, due to its main characteristic of integrate logic and DRAM layers. Hence, the HMC was chosen as the standard memory layout for our simulation environment. Moreover, due to the heterogeneity of PIM designs, the proposed simulator must allow to implement different types of PIM.

Figure 2 illustrates the most common PIMs that can be simulated by the present work. Figures 2a and 2b show the integration of two types of PIMs in the logic layer provided by 3D-stacked HMC module. It is possible to notice that the **2a** type consists of a complete CPU system comprising a processor and traditional memory hierarchy. On the other hand, **2b** type comprises a simple set of Functional Units (FUs) and a set of register files. A PIM that avoids traditional cache hierarchy is presented in Figure 2c. This type of PIM can be seen as a traditional accelerator. The PIM shown in Figure 2d is placed as close as possible to the memory array, or even along with the memory cells. This design is applied on different memory technologies, such as DRAM and Resistive RAM (ReRAM).

To support the simulation of the above mentioned PIM designs, the GEM5 simulator has been modified to include Instruction Set Architecture (ISA) extension, offloading, virtual address translation and data coherence mechanisms, just to list some of the requirements of different PIM designs.

4.1.1 ***HMC Modeling:*** Each type of PIM presented in Figure 2 requires different resources from memory systems.

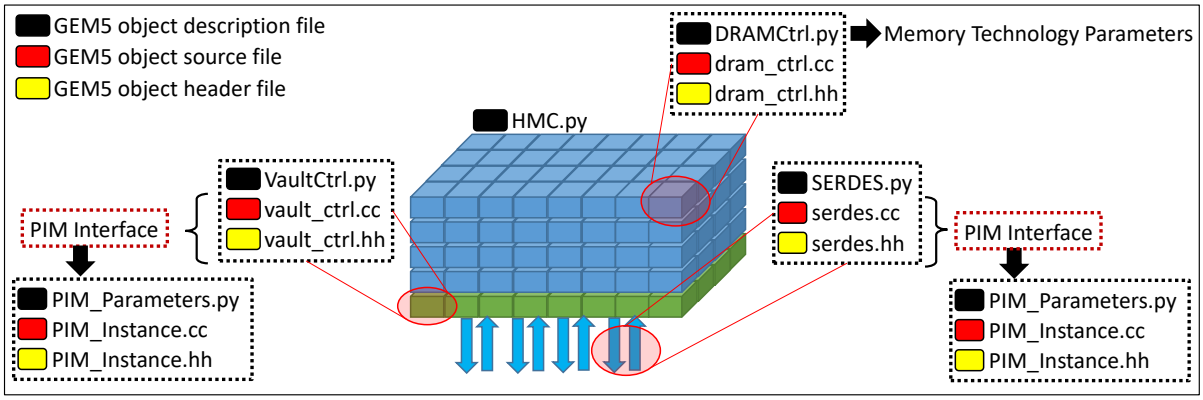

Fig. 3. Overview of the modified GEM5's HMC files

The PIMs illustrated by Figures 2a and 2b lies within 3D-staked HMC *vault* controllers. These types of PIM allow independent instances along all *vaults* making possible parallel and concurrent memory accesses, as presented in [2, 17, 51] and [47, 49, 55]. Therefore, to support the simulation of such designs, a new HMC representation comprising 32 independent *vault* controllers, 8 DRAM layers, and a low-level interleaving was modeled. Hence, our model can properly reproduce the HMC parallelism between memory *vaults* and banks. A suitable hierarchy of crossbars has been implemented to enable the correct connection between *links* and *vaults*, which is called quadrant in HMC specification [50]. These crossbars are also responsible for the communication between *vaults*, which allows the rapid exchange of data between different instances of the PIM.

PIMs represented by Figure 2c do not require modifications in main memory, since this type of design is seen as a typical accelerator [6]. However, the correct representation of the *links* is important to properly simulate the behavior of the experimented mechanism.

A different requirement is made by the type illustrated in Figure 2d. In this case, the PIM unit is placed along the DRAM array [13, 39, 58], which requires a different approach, since representing each cell individually would be prohibitive in terms of simulation time. Thus, we reuse the units shown in Figure 2b to simulate this type of PIM. By adjusting specific timings in different parts of the circuits (buses, commands, etc) it is possible to simulate groups of cells being accessed. This class is generally limited to bulk bitwise operations, although offloading, virtual address translation and data coherence mechanism are needed.

Figure 3 depicts the main files that have been modified on GEM5's HMC representation. Also, it highlights the *interface* point between the PIMs and memory module (and its main components).

4.1.2 ***Host Processor Modifications and PIM:***. PIM logic implemented with existing processor cores, such as the ones presented in Figures 2a and 2c, do not require modifications on a host processor. Also, all communication among host and accelerator or between instances of PIMs are managed by the programmer via software, such as OpenMP, MPI and special libraries.

On the other hand, PIM logics centered on simple processing units, such as those shown in Figures 2b and 2d, are dependent on a host. This class of PIM requires a host processor to dispatch PIM instructions or to perfom control-flow operations due to the fine-grain control of their logic. One approach to seamlessly

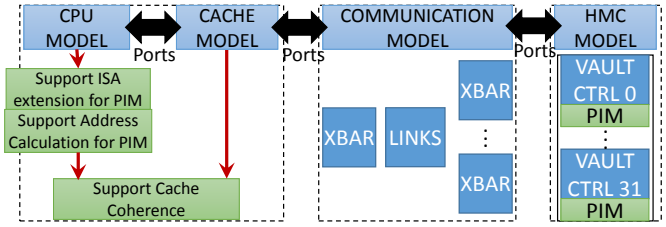

Fig. 4. Overview of the modules in the PIM Simulator

offload PIM instructions from the host processor to the PIM device is to make the host CPU compatible with a new PIM ISA. Thus, the machine code is composed of a *hybrid code*, which is detailed in Section 4.2.

The *hybrid code* is also applicable for native HMC commands [28] and some designs found in the literature [46, 49, 55] with the cost of developing a new ISA extension. The Reconfigurable Vector Unit (RVU) PIM design [55] was chosen to illustrate the implementation of this type of PIM. The RVU design extends the Advanced Vector Extensions (AVX)-512 ISA by allowing the computation of huge vector operands in memory through its large Vector Processor Units (VPUs), which provides the ability to exploit the internal bandwidth available in 3D-stacked memories. RVU PIM allows vector operands from 16 Bytes to 8192 Bytes [40, 55], while it has a VPU of 256 Bytes and a register file of the same size instantiated within each *vault* of the HMC.

Figure 4 illustrates the most important modifications made on GEM5 to support this type of PIM. Firstly, the host CPU was upgraded to support the PIM ISA extension. The processor *decoder* was updated to include the decoding of PIM instructions. Also, as RVU extends the AVX-512 ISA, the GEM5's X86 processor was also upgraded to support AVX-512 accordingly to [14, 15, 19]. Although RVU uses the same addressing modes present in AVX-512, it requires modifications on the Address Generation Unit (AGU) to calculate physical addresses of memory operands that occupy more than one virtual page of the Operating System (OS). Secondly, the *Load Store Unit (LSU)* module was modified to support the dispatching mechanism of PIM instruction with varied requirements. The *LSU* module also manages and performs cache coherence by flushing cache blocks when required to keep data consistency. The RVU ISA allows instruction with memory operand, immediate or host register operand as source to be operated on the PIM unit. It also defines synchronous instructions that expect a response from the PIM unit to be stored in a host register as destination. Using these types of instructions the programmer is allowed to exchange data between PIM registers and host register seamlessly.

4.2 Compiling for PIM

Another known issue while simulating state-of-the-art architectures is how to generate code for an experimental design. PIM types that use traditional programmable cores (Figures 2a and 2c) are able to take advantage of existing compilers, legacy libraries and programming models. As aforementioned, the data sharing among all processing units is done via specialized libraries [17, 26, 51, 59], which demands programming efforts.

New PIM architectures, such as native HMC PIM and the ones presented in Figures 2b and 2d, require a different solution. The *hybrid code* style can be a solution that allows the automatic offloading of instructions directly from host to PIM, as presented in Section 4.1.2. The *hybrid code* style consists of a mixed host and PIM instruction, which requires intrinsic collaboration between host and accelerator at

```
mov     r10, rdx
xor     ecx, ecx
RVU_256B_LOAD_DWORD  RVU_3_R256B_1, pimword ptr [rsp + 1024]
RVU_256B_VPERM_DWORD RVU_3_R256B_1, RVU_3_R256B_1, RVU_3_R256B_0
RVU_256B_VADD_DWORD  RVU_3_R256B_0, RVU_3_R256B_0, RVU_3_R256B_1
RVU_256B_STORE_DWORD pimword ptr [rsp + 1536], RVU_3_R256B_0
mov     eax, dword ptr [rsi + 4*rcx + 16640]
imul    eax, r9d
add     eax, dword ptr [rsp + 1536]
mov     dword ptr [r10], eax
inc     rcx
```

Fig. 5. Example of Hybrid Code (X86 and RVU)

instruction level. Therefore, this approach requires a compiler capable of generating a *hybrid* machine code containing both host ISA and PIM ISA from the same source code.

Processin-in-Memory cOmpiler (PRIMO) [1] is designed for PIMs that have the necessity of insctructions offloading decision and generation being made by a compiler. Moreover, PRIMO has focus in the exploitation of VPUs present in these machines. PRIMO is developed using the Low Level Virtual Machine (LLVM) compiler tool [36]. The three main modules that comprise a modern compiler pipeline flow, such as LLVM, are called *Front-End*, *Middle-End* and *Back-End*. The Front-End takes as input the source code files and basically performs the *lexical analysis*, *syntactic analysis* and *semantic analysis*. Hence, the output generated by the Front-End is an Intermediate Representation (IR) block of code which is the first compiler's internal representation derived from the source. The Middle-End is responsible for applying to the IR code optimizations such as peephole optimizations, loop unrolling and vectorization. Finally, the Back-End translates the already optimized IR into architecture instructions, performs some specific architectural optimizations and generates the binary file to be executed further.

The PRIMO tool provides an generic implementation such that for any target PIM, the compiler can be extended based on the architecture specific features. The modifications done in LLVM pipeline flow by PRIMO are described as follows. For the Front-End module, any modification is needed since the actions performed in this stage are kept the same and they are independent for any IR or architecture. Middle-End is extended to support bigger vector widths and an offloading mechanism. The offloading technique is based on data locality and vector width being responsible to decide whether to execute the code on PIM. Additionally, the Middle-End has specific PIM hardware usage optimizations implemented. The Back-End is updated to support the PIM register bank and the new extended PIM ISA. Optimizations related to explore better *vault*, memory bank and VPUs level parallelism in the HMC device are also introduced in this module. Also, optimizations regarding communication among PIM instances, register allocation and architectural instruction translations are added in the Back-End. Finally, the binary file containing a hybrid mix of PIM and X86 instructions is built to be executed.

The code snippet presented on Figure 5 illustrates the generated code for the RVU mechanism [1]. It is possible to observe that the generated code contains both X86 and RVU instructions, showing the hybrid style code where both X86 and PIM instructions are fetched, decoded, and dispatched by a host processor (Section 4.1.2). Hence, supported by the host modifications mentioned in Section 4.1.2, each instruction is fetched, decoded, and locally computed if recognized as typical X86 instruction. Otherwise the instruction is prepared along host pipeline, computing addresses and flushes when required, and lastly offloaded to the PIM to be properly processed.

The compiler selects the most suitable instructions to offload to the PIM accelerator, trying to exploit all available resources. It is possible to notice on Figure 5 that instructions like *RVU_256B_VADD_DWORD*

can be seen as a group of four AVX-512 *ADD* instruction, and the load operation *RVU_256B_LOAD_DWORD* can load 256 Bytes of data from main memory at once to an internal RVU register. Also, from the code snippet, it can be seen the RVU registers nomenclatures, which shows that for each HMC *vault* a group of RVU register is available.

4.3 Technological Agnostic Capabilities

PIM architectures have been studied for decades. However, with the advancement of 3D-stacked technologies, PIM have regained attention as the 3D-stacked allows the mix of logic and memory (DRAM) on same chip. Furthermore, academical and industrial researches have present new memory technologies, such as ReRAM, Phase-Change Memory [12, 29, 60], as well as ways to use memory to achieve energy-efficient computing. However, creating a specific simulator for each technology is a time consuming task, mainly when the motivation is to evaluate different memory technologies. In this way, ReRAM, DRAM, Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM) read/write latencies and energy costs obtained by CACTI or other type of estimation can be embedded in GEM5, as already tested in previous work [42]. Then, the framework is easily extended to simulate different memory technologies and PIM mechanisms by configuring the timings to represent the required behavior. In the same way, the memory controller can be improved to support analogical operations and simulate addition and multiplications inside the memory crossbar [29].

5 CASE STUDY

With the spread of sensing devices collecting huge amounts of data, and the requirement that each small device be able to efficiently support complex applications, a new class of hardware devices have emerged. Moreover, the ever increasing amount of data leads to a processing hungry fashion applications, which urges processing power.

The CPS and IoT devices represents this new class, being applied from the Industry 4.0, critical medical systems, home devices, autonomous vehicles, and military drones to toys, mobile smart devices, academic studies, research environment monitoring, etc.

Nowadays, one of the most critical, complex and demanded application applies deep learning algorithms in CPS and IoT devices. Object detection, pattern and speech recognition, context-aware recommender systems are common examples. A most complex example are shown by computer vision in autonomous vehicles, where *LiDAR*, radars, and camera sensors generate streaming videos that must be processed in a constrained time. The example can become more power hungry with the addition of Neural Network (NN) based applications such as pedestrian detection, transit and traffic recognition, which requires huge processing power on large volume of data.

Since this is a challenging scenario, the chosen case study consists in coupling several camera sensors to a PIM and efficiently implement an image recognition software for this architecture by using our proposed framework. As application domain example, we have chosen to use the algorithm provided by [52] in our experiments. Their algorithm, called YOLO9000, is a state-of-the-art real-time object detection algorithm faster than previous algorithms of the same class while providing enough accuracy.

5.1 Proposed Architecture

The main challenge on embedded devices lies on allow high processing power and energy efficiency together. CPS and IoT devices are expected to leverage this behavior. To support this idea, the proposed architecture avoids traditional complex I/O interfaces by taking advantage of available resources present on

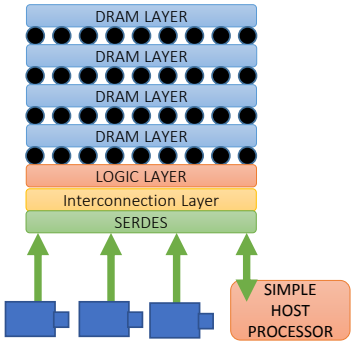

Fig. 6. Small Host Processor + HMC + Sensors

HMC module. Figure 6 illustrates a generic solution for this application domain. The proposed architecture can be applied to any previously mentioned state-of-the-art architecture (Figure 2 in Section 4.1).

It is composed of a PIM, a host processor, and sensors directly connected to the 3D-stacked memory. In the case of HMC module, the sensors are connected directly to the memory via high-speed serial links, the same used to connect host CPUs and memory device [28]. Each sensor is configured with a physical base address, and all write operations from these sensors are based on that address. This way, as each sensor has its own range of addresses, the sensors can trigger write operations without the need of address translation, while ensure data consistence. Also, the proposed architecture allow the concurrent writing behavior with no penalties, which means that many sensors can write data directly to the memory taking advantage of the inherent parallelism and the high bandwidth in an efficient fashion.

5.2 Simulating Approach

In the proposed architecture, the PIM instructions are decoded and dispatched by a small X86 host processor. The cache is needed to store host-side and PIM instructions, but most of the YOLO kernel (heavy computation) is processed using PIM instructions, and the memory requests are made only inside the HMC device. The host CPU is mostly used in branch instructions of YOLO kernel and also other parts of the software responsible for scheduling operations on the range of cameras, which allow us to run frames from different sources simultaneously.

Traffic generators are attached to high-speed serial links to simulate sensors storing data. As the workload from different sources can be easily explored by Thread Level Parallelism (TLP), we consider a multi-issue processor with 32 independent lanes. To do so, we have employed a simple static scheduler and an interconnection structure to exploit concurrent executions. This static scheduler is used to map the different sensors to our device and allow us to run frames from different sources simultaneously. Then, each sensor is seen as a thread with reserved memory space, where the PIM instructions modify the data over a particular memory region.

6 EXPERIMENTAL APPLICATION

In this section, we describe the simulation setup and methodology employed, and the results of our experiments.

Table 1. Baseline and design system configuration.

Baseline x86 Processor X86-based out-of-order multi-core processor; 4 cores@3GHz; IL1 64KB + DL1 64KB; L2 256kB; L3 Cache 8MB; 8 issues - SSE and AVX-512 Instruction Set Capable; Power Consumption 40W@32nm;
ARM-based PIM ARM-based in-order single-core processor; 16 PIM Instances - 1 core@2GHz; IL1 64kB + DL1 64kB; no L2; no L3; 6 issues - Neon-128 Instruction Set Capable;
X86-based PIM X86-based out-of-order single-core processor; 16 PIM Instances - 1 core@2GHz; IL1 64kB + DL1 64kB; no L2; no L3; 8 issues - AVX-512 Instruction Set Capable;
RVU PIM 1GHz; 32 Independent Functional Units; Integer and Floating-Point Capable; Vectorial Operations up to 256 Bytes per Functional Units; 32 Independent Register Bank of 8x256Bytes each; Latency (cycles): 1-alu, 3-mul. and 20-div. integer units; 5-alu, 5-mul. and 20-div. floating-point units; Interconnection between vaults: 5 cycles latency; X86-based in-order Host Processor - 2GHz; Single Core; IL1 64KB + DL1 64KB; no L2; no L3; Power Consumption 4W@32nm + 16W@32nm [38, 40];
HMC Module HMC version 2.0 specification; Total DRAM Size 8GBytes - 8 Layers - 8Gbit per layer 32 Vaults - 16 Banks per Vault; 4 high speed Serial Links; Energy Consumption 10pJ/bit [30, 40]
DRAM Timings CAS, RP, RCD, RAS, CWD latency (9-9-9-24-7 cycles);
ReRAM Timings [44] Write, Read latency (0.25-0.25 ns per cell); Considering 16MB per bank; Energy Consumption 10^{-5}pJ/cell [42];

6.1 Experimental Setup

Firstly, in order to compare the simulation time, and to show the generality of the proposed framework, this section shows a comparison between a single-core out-of-order Intel processor and a multi-core version of the same processor, both coupled with the developed HMC module. Moreover, to show the PIM simulation capabilities, two types of PIM are implemented, illustrated in Figure 2a ([55])and Figure 2b (inspired by [2, 17]). Since the RVU PIM requires instruction offloading, it is coupled with a single-core processor host for instruction offloading, and all PIMs are integrated within the logic layer provided by the HMC, which is configured with 32 *vaults* to accommodate up to 32 PIM instances.

Secondly, the presented framework is used to simulate the IoT design as case study shown in the Section 5. The experiment consists of connecting several video-camera sensors (IoT devices) to the same HMC module, and the image recognition application is applied to each single frame. A PIM is responsible for efficiently compute the data.

The baseline system, the PIM configurations, power, and energy consumption constraints are described in the Table 1. For all tests, the system used to run the simulator is comprised of a processor Intel i7-4700 and 16GB of main memory.

6.2 Simulation time

In order to evaluate the performance of the proposed GEM5 modifications, Figure 7 shows the simulation time comparison between the baseline processor and the different multi-core and PIM implementations running 3 simple kernels - *vecsum*, *stencil-2d*, and *matrixmul*. All results are normalized to the simulation

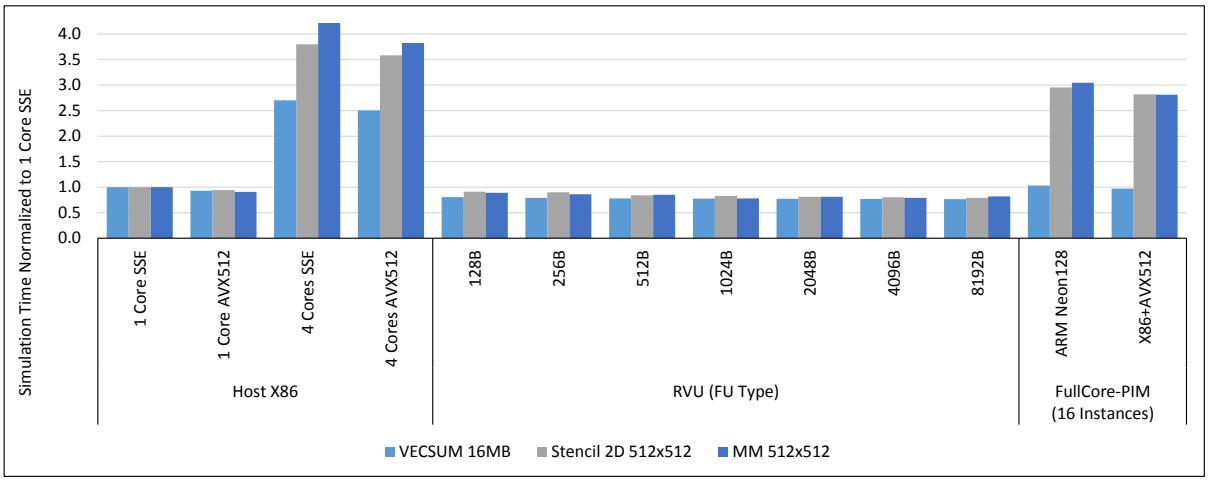

Fig. 7. Simulation time for a 4-core processor, different RVU operand sizes [55], an ARM-based PIM, and a X86-based PIM with its 16 core Instances

time of the single core SSE scenario, and in all cases the HMC module is used, accordingly to Table 1 parameters.

From Figure 7 it is possible to observe that the X86 multi-core scenario requires bigger simulation time than the single-core approach. This occurs because the GEM5 is a sequential simulator, improving the execution time according to the amount of hardware simulated. On the other hand, in case of full-cores PIM approach, they are implemented using simpler cores, with significantly reduced cache levels, which improves simulation time. Furthermore, another important point that must be observed is the number of simulated instructions, which is proportional to the available operand sizes. As the simulated PIM and AVX-512 processors reduce the number of instructions executed (due to the improvement on vector capabilities), the simulation time is drastically reduced. Also, it is important to notice that the RVU PIM is able to operate through different operand sizes per instructions, thus operating from 128 Bytes to 8192 Bytes of data at once, which further reduces the simulation time (and performance as presented in the Section 6.3).

6.3 Image Recognition Experimentation for IoT Devices

Supported by the present framework, this section evaluates the behavior, operations, and performance of the proposed architecture presented in Section 5. Moreover, different General Purpose Processor (GPP) and PIM designs are evaluated, in order to show the generality of the simulating environment.

Following the case study, Figure 8 illustrates how several IoT devices can be connected to the proposed system. Three *Links* provided by HMC are used as input buses from IoT sensors, while one is reserved to the host processor that is responsible for triggering instructions to the RVU instances. The *YOLO* application is computed by the experimented designs, whose data are serviced by traffic generators representing cameras.

6.3.1 Performance Evaluation. As aforementioned, the *Yolo* application represents an important modern application class. Moreover, the Convolutional Neural Nertwork (CNN) algorithm is widely used in different image recognition applications, and we show that the PIM approach is suitable for this class of

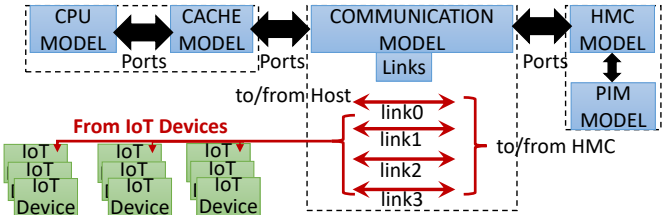

Fig. 8. Several IoT connected to native HMC links

application in the present experiment. In order to exploit the proposed architecture, we connect an IoT device to a *link*, and we tune up the source Frames per Second (FPS). The goal in this experiment is to find the FPS limit that each evaluated architecture can achieve, while continuously, at each frame, execute the image recognition application. The goal is to find the maximum FPS that each evaluated architecture can achieve, while continuously, at each frame, execute the image recognition application.

Figure 9 presents the performance achieved by the evaluated architectures running the *YOLO* application. It shows the maximum number of images that the traditional processors and the implemented PIMs can compute per second (FPS). For all cases, it is shown the simulations coupled with HMC module using DRAM timings and ReRAM timings. Also, in Figure 9, the stacked bars highlight the difference of behavior on the application caused by each architecture and memory technology.

Firstly, to analyze the impact of different processing architectures, it is possible to notice that the traditional processor configured with 4 cores coupled to HMC memory can achieve 8 FPS, while on the other extreme, the RVU coupled with same HMC can achieve 64 FPS. One can observe that the RVU approach, supported by its large vector operands (up to 8kB), takes advantage of the streaming portion of the code (*GEMM + Memory* access). However, RVU uses its host processor to compute *branch* instructions, and sequential scalar operations and memory access, which is shown by the *Others + Memory*.

Figure 9 also shows the results achieved by the ARM-based and X86-AVX512-based PIMs. Both are implemented with 16 instances (16 PIM-cores distributed along the HMC *vaults*), and they achieve 39 and 48 FPS, respectively. Moreover, both cases show that the main efforts are put on the computation of the

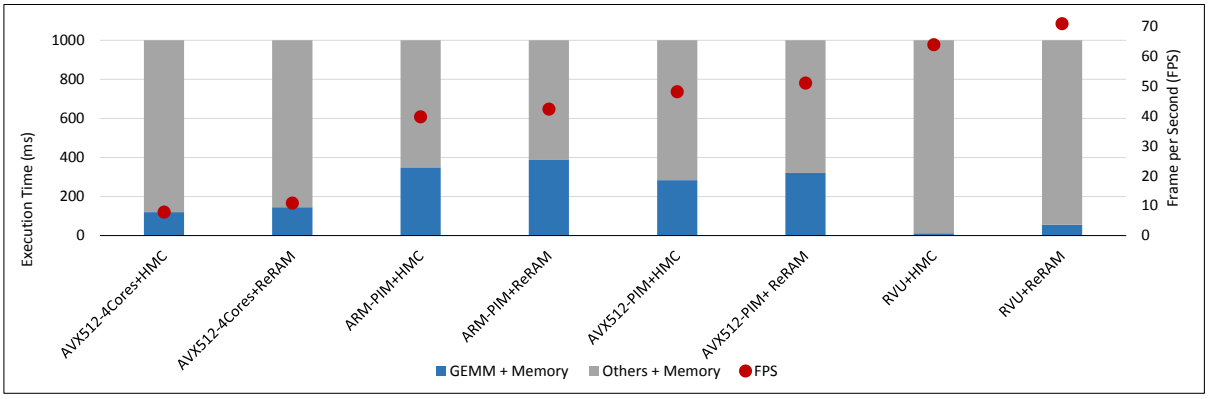

Fig. 9. FPS results and operation time distribution

GEMM algorithm. This happens because their implementation does not count on bigger cache memories, and neither counts of large vector operand capabilities, although they are able to take advantage of near data computing.

The reduced access time results presented by ReRAM can be observed Figure 9. By changing from DRAM to ReRAM, all systems experience a performance improvement, however it occurs in different proportions for each evaluated system. The FPS achieved by the 4 cores GPP improved from 8 to 11 FPS, while the improvement presented by the RVU jumped from 64 to 71 FPS. In case of GPP systems, regardless the use of cache memories (8MB of LLC), the streaming behavior of the application requires several memory access (64 Bytes per access), which shows the significance of the memory access latency. Differently, due to its large vector operands, RVU takes advantage of bandwidth more than latency reduction, which can be noticed by the slightly increased on the processing time for *GEMM* operations that now appears requiring a larger portion. This way, the multi-core GPP achieves 39% of performance improvement, while the RVU achieves 11%.

In case of GPP PIMs, similar to RVU, the near data processing behavior allows a better usage of the internal bandwidth. However, these designs present small vector capabilities that limits their performance. Therefore, by adopting ReRAM, the ARM-PIM and X86-PIM has increased from 39 to 42 FPS (6.5%), and from 48 to 51 FPS (6%), respectively.

6.3.2 Energy Consumption Evaluation. Modern applications of CPS and IoT devices are expected to compute an ever increasing data fashion. Moreover, the nature of such applications are moving towards a more complex and processing hungry requirements. However, as the processing power increases, energy consumption remains a major concern in these systems. Based on this and supported by McPAT [38] and CACTI [45], by implementing the hardware description in the present framework it is possible to estimate the energy consumption for the experimented designs. Figure10 summarizes the total and the distributed energy consumption for the GPP and RVU while processing the *YOLO* application for one frame.

It is noticeable that by replacing main memory (DRAM to ReRAM), the total energy consumption (red dot) decreases for all designs. This behavior is more pronounced on typical GPP, which is corroborated by the FPS performance improvement shown in Figure 9. As the GPP requires more memory accesses, the impact of main memory energy is bigger.

The simplest design implemented by the RVU, which applies small in-order processor, avoids complex large cache memories, and improves performance, reflects in the energy consumption. The improvement on energy consumption achieves 10× when compared against the 4-cores GPP system. Also, Figure 10

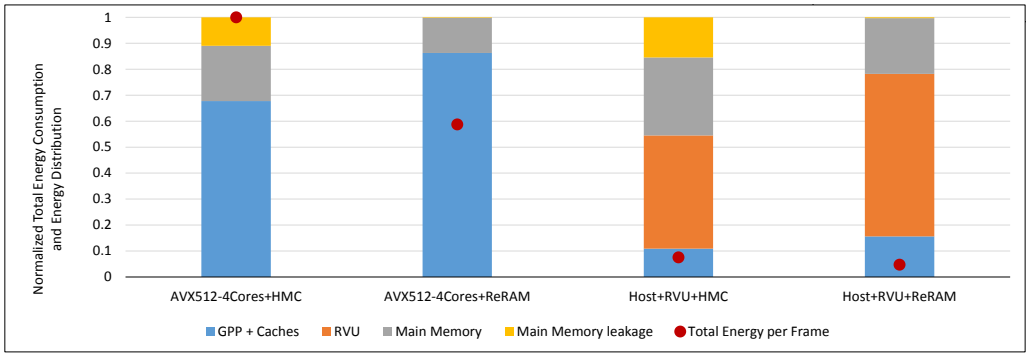

Fig. 10. Normalized energy consumption

shows that the RVU FUs dominates the energy consumption of the system, and per frame, it consumes 65% of the total energy couples with ReRAM.

7 CONCLUSIONS AND FUTURE WORK

This work presents a design framework based on the GEM5 environment and an LLVM-based compiler to simulate CPS and IoT devices. Our case study shows that, by connecting several sensors on the fast links present on the HMC module design, one can concurrently process several workload streams from different sources in the same PIM system by exploiting SIMD in machine learning algorithms, such as real-time object recognition. Moreover, this work shows that IoT devices can collect massive data, which requires substantial computational power that can be supplied by the state-of-the-art PIM mechanism. The efficient exploitation of the RVU PIM design is shown in the Section 6. As a proof of concept, we simulated different scenarios of our proposed architecture for IoT devices running *Yolo* application using DRAM and ReRAM memory technologies, which achieves 64 FPS and 71 FPS, respectively. The required simulation time is proportional to the original GEM5 simulation engine. Finally, the simulator can be easily extended to support new features to be evaluated in the design exploration of PIM architecture using the same compiler, and we have demonstrated that it can be modified to assess the effects of new organizations and technologies.

In order to further explore the proposed architecture, as future work a study presenting a large number of heterogeneous sensors can show the limits of the system. Also, in this scenario a scheduler to efficiently exploit the resources will be a challenge.

REFERENCES

[1] Hameeza Ahmed, Paulo C. Santos, João Paulo Lima, Rafael F. Moura, Marco A. Z. Alves, Luigi Carro, and Antonio C. S. Beck. 2019. A Compiler for Automatic Selection of Suitable Processing-in-Memory Instructions. In *Design, Automation and Test in Europe Conf. and Exhibition (DATE), 2019*.

[2] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2016. A scalable processing-in-memory accelerator for parallel graph processing. *ACM SIGARCH Computer Architecture News* 43, 3 (2016), 105–117.

[3] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. PIM-enabled Instructions: A Low-overhead, Locality-aware Processing-in-memory Architecture. In *Int. Symp. on Computer Architecture (ISCA)*.

[4] Omar Y Al-Jarrah, Paul D Yoo, Sami Muhaidat, George K Karagiannidis, and Kamal Taha. 2015. Efficient machine learning for big data: A review. *Big Data Research* 2, 3 (2015), 87–93.

[5] M. A. Z. Alves, M. Diener, P. C. Santos, and L. Carro. 2016. Large vector extensions inside the HMC. In *Conf. on Design, Automation & Test in Europe*.

[6] Hadi Asghari-Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim. 2016. Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 50.

[7] Erfan Azarkhish, Davide Rossi, Igor Loi, and Luca Benini. 2016. A Case for Near Memory gem5 Inside the Smart Memory Cube. In *Workshop on Emerging Memory Solutions, 2016*.

[8] Erfan Azarkhish, Davide Rossi, Igor Loi, and Luca Benini. 2016. Design and evaluation of a processing-in-memory architecture for the smart memory cube. In *International Conference on Architecture of Computing Systems*. Springer, 19–31.

[9] Erfan Azarkhish, Davide Rossi, Igor Loi, and Luca Benini. 2017. Neurostream: Scalable and Energy Efficient Deep Learning with Smart Memory Cubes. *arXiv preprint arXiv:1701.06420* (2017).

[10] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, et al. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7.

[11] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[12] Geoffrey W Burr, Matthew J Brightsky, Abu Sebastian, Huai-Yu Cheng, et al. 2016. Recent progress in phase-change memory technology. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 6, 2 (2016), 146–162.

[13] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 27–39.

[14] Intel Corportation. 2015. Intel Architecture Instruction Set Extensions Programming Reference. *Intel Corportation* (2015).

[15] Intel Corportation. 2016. Intel 64 and IA-32 architectures software developer's manual. *Intel Corportation* (2016).

[16] W Rhett Davis, John Wilson, Stephen Mick, Jian Xu, Hao Hua, Christopher Mineo, Ambarish M Sule, Michael Steer, and Paul D Franzon. 2005. Demystifying 3D ICs: The pros and cons of going vertical. *IEEE Design & Test of Computers* 22, 6 (2005), 498–510.

[17] Mario Drumond, Alexandros Daglis, Nooshin Mirzadeh, Dmitrii Ustiugov, Javier Picorel, Babak Falsafi, Boris Grot, and Dionisios Pnevmatikatos. 2017. The mondrian data engine. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, 639–651.

[18] A. Farmahini-Farahani, J. H. Ahn, K. Compton, and N. S. Kim. 2014. DRAMA: an architecture for accelerated processing near memory. *Computer Architecture Letters* 99 (2014).

[19] Agner Fog et al. 2011. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs. *Copenhagen University College of Engineering* 97 (2011), 114.

[20] Di Gao, Tianhao Shen, and Cheng Zhuo. 2018. A design framework for processing-in-memory accelerator. In *Proceedings of the 20th System Level Interconnect Prediction Workshop*. ACM.

[21] Mingyu Gao and Christos Kozyrakis. 2016. HRL: Efficient and flexible reconfigurable logic for near-data processing. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. Ieee, 126–137.

[22] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *Int. Conf. on Architectural Support for Programming Languages and Operating Systems*. ACM, 751–764.

[23] Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* (2013).

[24] Ramyad Hadidi, Lifeng Nai, Hyojong Kim, and Hyesoon Kim. 2017. CAIRO: A Compiler-Assisted Technique for Enabling Instruction-Level Offloading of Processing-In-Memory. *ACM Transactions on Architecture and Code Optimization (TACO)* 14, 4 (2017), 48.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*. 770–778.

[26] Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W Keckler. 2016. Transparent offloading and mapping (TOM): Enabling programmer-transparent near-data processing in GPU systems. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 204–216.

[27] Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu. 2016. Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation. In *Int. Conf. on Computer Design (ICCD)*.

[28] Hybrid Memory Cube Consortium. 2013. Hybrid Memory Cube Specification Rev. 2.0. http://www.hybridmemorycube.org/.

[29] Mohsen Imani, Saransh Gupta, and Tajana Rosing. 2017. Ultra-efficient processing in-memory for data intensive applications. In *Annual Design Automation Conference*. ACM.

[30] Joe Jeddeloh and Brent Keeth. 2012. Hybrid memory cube new DRAM architecture increases density and performance. In *VLSI Technology (VLSIT), 2012 Symposium on*. IEEE, 87–88.

[31] Andrej Karpathy. 2015. The unreasonable effectiveness of recurrent neural networks. *Andrej Karpathy blog* (2015).

[32] Chad D Kersey, Hyesoon Kim, and Sudhakar Yalamanchili. 2017. Lightweight SIMT core designs for intelligent 3D stacked DRAM. In *Proceedings of the International Symposium on Memory Systems*. ACM, 49–59.

[33] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. 2016. Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory. *Int. Symp. on Computer Architecture, ISCA* (2016).

[34] Michael B Kleiner, Stefan A Kuhn, and Werner Weber. 1995. Performance improvement of the memory hierarchy of RISC-systems by application of 3-D-technology. In *Electronic Components and Technology Conference, 1995. Proceedings., 45th*. IEEE, 645–655.

[35] Peter M Kogge, Jay B Brockman, Thomas Sterling, and Guang Gao. 1997. Processing in memory: Chips to petaflops. In *Workshop on Mixing Logic and DRAM: Chips that Compute and Remember at ISCA*, Vol. 97. Citeseer.

[36] Chris Lattner and Vikram Adve. 2004. LLVM: A compilation framework for lifelong program analysis & transformation. In *Int. Symp. on Code Generation and Optimization: Feedback-directed and Runtime optimization*. IEEE Computer Society, 75.

[37] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin, J. H. Cho, K. H. Kwon, M. J. Kim, J. Lee, K. W. Park, B. Chung, and S. Hong. 2014. 25.2 A 1.2V 8Gb 8-channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 432–433.

[38] Sheng Li, Jung Ho Ahn, Richard D Strong, et al. 2013. The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing. *Transactions on Architecture and Code Optimization* 10, 1 (2013), 5.

[39] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. 2016. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 173.

[40] João Paulo Lima, Paulo C. Santos, Marco A. Z. Alves, Antonio C. S. Beck, and Luigi Carro. 2018. Design space exploration for PIM architectures in 3D-stacked memories. In *Proceedings of the Computing Frontiers Conference*. ACM.

[41] Christianto C Liu, Ilya Ganusov, Martin Burtscher, and Sandip Tiwari. 2005. Bridging the processor-memory performance gapwith 3D IC technology. *IEEE Design & Test of Computers* 6 (2005), 556–564.

[42] Manqing Mao, Yu Cao, Shimeng Yu, and Chaitali Chakrabarti. 2016. Optimizing latency, energy, and reliability of 1T1R ReRAM through cross-layer techniques. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 6, 3 (2016), 352–363.

[43] Kyungmin Cho Jaemoo Lee Jongkyeong Park Minchul Kim, Jeonghyun Lee. [n. d.]. LG Electronics 16-cameras - Patent US010135963.

[44] Amir Morad, Leonid Yavits, Shahar Kvatinsky, and Ran Ginosar. 2016. Resistive GP-SIMD processing-in-memory. *ACM Transactions on Architecture and Code Optimization (TACO)* 12, 4 (2016), 57.

[45] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. 2009. CACTI 6.0: A tool to model large caches. *HP laboratories* (2009), 22–31.

[46] Lifeng Nai, Ramyad Hadidi, Jaewoong Sim, Hyojong Kim, et al. 2017. Graphpim: Enabling instruction-level pim offloading in graph computing frameworks. In *Int. Symp. on High Performance Computer Architecture (HPCA)*. IEEE, 457–468.

[47] R. Nair, S. F. Antao, C. Bertolli, P. Bose, J. R. Brunheroto, T. Chen, C. Y. Cher, et al. 2015. Active Memory Cube: A processing-in-memory architecture for exascale systems. *IBM Journal of Research and Development* 59, 2/3 (March 2015), 17:1–17:14.

[48] Geraldo F Oliveira, Paulo C Santos, Marco AZ Alves, and Luigi Carro. 2017. A Generic Processing in Memory Cycle Accurate Simulator under Hybrid Memory Cube Architecture. In *Int. Conf. on Embedded Computer Systems: Architectures, Modeling and Simulation*.

[49] Geraldo F Oliveira, Paulo C Santos, Marco AZ Alves, and Luigi Carro. 2017. NIM: An HMC-Based Machine for Neuron Computation. In *International Symposium on Applied Reconfigurable Computing*. Springer, Cham, 28–35.

[50] J Thomas Pawlowski. 2011. Hybrid memory cube (HMC). In *2011 IEEE Hot Chips 23 Symposium (HCS)*. IEEE, 1–24.

[51] Seth H Pugsley, Jeffrey Jestes, Huihui Zhang, Rajeev Balasubramonian, Vijayalakshmi Srinivasan, Alper Buyuktosunoglu, Al Davis, and Feifei Li. 2014. NDC: Analyzing the impact of 3D-stacked memory+ logic devices on MapReduce workloads. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 190–200.

[52] Joseph Redmon and Ali Farhadi. 2016. YOLO9000: Better, Faster, Stronger. *arXiv preprint arXiv:1612.08242* (2016).

[53] Mengye Ren, Ryan Kiros, and Richard Zemel. 2015. Exploring models and data for image question answering. In *Advances in Neural Information Processing Systems*. 2953–2961.

[54] Paulo C. Santos, Joao Paulo C. de Lima, Rafael Fão de Moura, Hameeza Ahmed, Marco Antonio Zanata Alves, Antonio C. S. Beck, and Luigi Carro. 2018. Exploring IoT platform with technologically agnostic processing-in-memory framework. In *INTESA@ESWEEK*.

[55] Paulo C Santos, Geraldo F Oliveira, Diego G Tomé, Marco AZ Alves, Eduardo C Almeida, and Luigi Carro. 2017. Operand size reconfiguration for big data processing in memory. In *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 710–715.

[56] R. R. Schaller. 1997. Moore's law: past, present and future. *IEEE Spectrum* 34, 6 (Jun 1997), 52–59.

[57] Marko Scrbak, Mahzabeen Islam, Krishna M Kavi, Mike Ignatowski, and Nuwan Jayasena. 2017. Exploring the Processing-in-Memory design space. *Journal of Systems Architecture* 75 (2017), 59–67.

[58] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. 2017. Ambit: In-memory accelerator for bulk bitwise operations

using commodity DRAM technology. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 273–287.

[59] Zehra Sura, Arpith Jacob, Tong Chen, Bryan Rosenburg, Olivier Sallenave, Carlo Bertolli, Samuel Antao, Jose Brunheroto, Yoonho Park, Kevin O'Brien, et al. 2015. Data access optimization in a processing-in-memory system. In *Proceedings of the 12th ACM International Conference on Computing Frontiers*. ACM, 6.

[60] Yuan Xie. 2011. Modeling, architecture, and applications for emerging memory technologies. *IEEE Design & Test of Computers* 28, 1 (2011), 44–51.

[61] Lifan Xu, Dong Ping Zhang, and Nuwan Jayasena. 2015. Scaling Deep Learning on Multiple In-Memory Processors. *WoNDP: 3rd Workshop on Near-Data Processing* (2015).

[62] Sheng Xu, Xiaoming Chen, Ying Wang, Yinhe Han, Xuehai Qian, and Xiaowei Li. 2018. PIMSim: A Flexible and Detailed Processing-in-Memory Simulator. *IEEE Computer Architecture Letters* (2018).

[63] Xu Yang, Yumin Hou, and Hu He. 2019. A Processing-in-Memory Architecture Programming Paradigm for Wireless Internet-of-Things Applications. *Sensors* 19, 1 (2019), 140.

[64] Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L Greathouse, Lifan Xu, and Michael Ignatowski. 2014. TOP-PIM: throughput-oriented programmable processing in memory. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*. ACM, 85–98.

[65] Richard Zhang, Phillip Isola, and Alexei A Efros. 2016. Colorful image colorization. In *European Conference on Computer Vision*. Springer, 649–666.

[66] Qiuling Zhu, Berkin Akin, H Ekin Sumbul, Fazle Sadi, James C Hoe, Larry Pileggi, and Franz Franchetti. 2013. A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing. In *3D Systems Integration Conference (3DIC), 2013 IEEE International*. IEEE, 1–7.


- MRE