Exploiting Reconfigurable Vector Processing for Energy-Efficient Computation in 3D-Stacked Memories

João Paulo C. de Lima[1(✉)], Paulo C. Santos[1], Rafael F. de Moura[1], Marco A. Z. Alves[2], Antonio C. S. Beck[1], and Luigi Carro[1]

[1] Informatics Institute, Federal University of Rio Grande do Sul, Porto Alegre, Brazil
jpclima@inf.ufrgs.br

[2] Department of Informatics, Federal University of Paraná, Curitiba, Brazil

Abstract. Although Processing-in-Memory (PIM) architectures have helped to reduce the effect of the memory wall, the logic placed inside 3D-memories still faces the large disparity between DRAM and CMOS logic operations. Thereby, for a broad range of emerging data-intensive applications, the Functional Units (FUs) are usually underutilized, especially when the application presents poor temporal-locality. As applications demand irregular processing requirements on the different parts of their execution, this behavior can be used to reconfigure energy-reduction techniques, either by scaling frequency or by power-gating functional units. In this paper, we present the application-dependable characteristics that enable dynamic usage of energy-reduction techniques without performance degradation for highly constrained PIM designs. The experimental results show that the exploration of a reconfiguration mechanism can improve PIM system energy efficiency by 5× and also can effectively benefit both memory-intensive and compute-intensive applications.

Keywords: Processing in Memory · Reconfigurable vector architectures · Energy-efficiency

1 Introduction

The 3D-stacking process has emerged as a solution for mitigating the memory-wall problem. In recent years, the emergence and feasibility of 3D-stacking technology have opened up opportunities in both architectural and chip design fields. Supported by these new trends, Processing-in-Memory (PIM) concept has emerged as a prominent approach to improve performance and reduce the energy of modern systems. This approach keeps closer processing and data by taking advantage of logic layer available on 3D-stacked memories to compute data directly in the memory device. Nevertheless, the design of 3D-stacked PIM devices still faces challenges related to costs, retention characteristics, and business decisions. The main issue resides on the thermal dissipation challenges that

C. Hochberger et al. (Eds.): ARC 2019, LNCS 11444, pp. 262–276, 2019.
https://doi.org/10.1007/978-3-030-17227-5_19

happen when stacking Dynamic Random Access Memory (DRAM) layers on top of processor layers. [1]. A second issue involves customizing the DRAM dies for each processor chip, which introduces design and supply-chain complexity that would increase the overall manufacturing cost [2].

Although PIM architectures have reduced the effects of the memory wall, the processing logic placed inside 3D-memories still faces the disparity of latencies between DRAM and CMOS logic operation. The average latency for DRAM access is typically tens of times higher than the time of a Functional Unit (FU) operation under the same constraints, and even worse for the most recent technology nodes. This fact implies that the processing unit can potentially spend more time in idle mode and increase power density depending on the arithmetic intensity. The degree of utilization of FUs relies on data reuse and the computational intensity inherent from the workload.

Furthermore, the majority of the applications can present a mix of compute-bound and memory-bound behavior [3]. Thus, the variability of application's demand for processing power and the latency disparity between operations on DRAM and FUs can be used to reduce energy consumption, and also help with the inherited thermal dissipation problems of the 3D-stacked PIM architectures. To do so, a special mechanism must detect the fluctuations in the application needs for FU resources. Further, this mechanism must reconfigure the PIM architecture to dynamically match the current demand for processing power and keep the maximum memory bandwidth achievable by each application part based on some decision heuristic. A reconfiguration process should dynamically adapt the number of FUs or perform a frequency scaling operation ideally without performance penalties. Adjusting the number of working FUs implies that highly parallel and bigger operations can be split into smaller and sequential ones. Thus, both the idle time and the energy consumption of unused FUs could be minimized.

This paper opens up a discussion on the use of reconfigurability in vectorial PIM architectures to provide energy-efficiency and overcome technical issues, rather than limiting the effects of reconfiguration to performance-oriented goals. The main contributions of this paper are:

1. The use of reconfigurability to minimize thermal power dissipation challenges in 3D-stacked PIM architectures.
2. The identification of application characteristics to match processing power to the maximum bandwidth achievable by each application.
3. A reconfigurable mechanism for dynamically reducing the number of active FUs as the application demands, which varies the processing power of PIM logic and finds a near-optimal point to the energy consumption.

The rest of this paper is organized as follows: in Sect. 2 a general overview of 3D-stacked PIM is presented, as well as its constraints and feasibility challenges are discussed. In Sect. 3, hardware and software aspects of reconfiguration on vector processors are discussed, and some examples are shown to highlight the benefits of reconfiguration. The experimental setup and methods used to validate the proposed method are described in Sect. 4 and the results are presented in

Sect. 4.3. Finally, some related works are listed in Sect. 5 and final considerations are made in Sect. 6.

2 Background

In this section, a general overview of fundamental concepts related to 3D-stacked memories and Processing-in-Memory (PIM) architectures are presented. Next, a brief discussion about 3D-PIM architectures design feasibility and constraints is promoted to situate the proposed approach realm.

2.1 3D-Stacked Processing-in-Memory

By connecting Dynamic Random Access Memory (DRAM) memory dies stacked on top of a logic layer using dense Through-Silicon Via (TSV), high-density 3D memories can provide higher capacity, bandwidth, and lower access latencies compared to traditional DRAM modules. The most diffused and recent examples of 3D-memory usage in the industry are the Micron's Hybrid Memory Cube (HMC) [4], and AMD/Hynix's High Bandwidth Memory (HBM) [5]. Figure 1 illustrates the internal organization of a generic 3D-stacked memory. For HMC and HBM, the 3D memory layout is composed of several DRAM layers, each one containing multiple banks.

The stacked arrangement of DRAM layers is split vertically into *vaults*. Each vault comprises a region of DRAM layers and a logic layer connected by an independent group of TSV and controlled by a vault controller. Each vault controller manages its own DRAM banks independently. Thus, it is possible to operate on a 3D-memory with both vault-level and bank-level parallelism. Following the last HMC specification [4], it can be seen that the memory module can have either four or eight DRAM dies and one logic layer all stacked within a memory cube. The cube has 32 vaults with their respective vault controllers. Each vault controller can manage independently 16 memory banks. The HMC can achieve up to 320 GB/s of bandwidth distributed along four serial link interfaces.

In addition to the emerging of 3D-stacked memories, the increasing demand for computational resources by data-intensive applications leveraged and reintroduced the PIM research field. As the usage of PIM devices alleviates the memory bottleneck, the natural application domain covers current big-data processing. Also, PIM reduces energy consumption and accelerates applications execution time avoiding the data movements back and forth along the memory hierarchy. Moreover, PIM can exploit both the high memory bandwidth, more massive Data Level Parallelism (DLP) and vault-level parallelism when coupled with 3D-stacked memories.

There are several architectural PIM design approaches in the literature. For commodity, and according to [1], they can be classified into two main categories based on the central processing element type present in the logic layer: General Purpose Processor (GPP)-like cores and dedicated logic circuits. The former group adopts the replication of conventional GPPs into the logic layer. This PIM

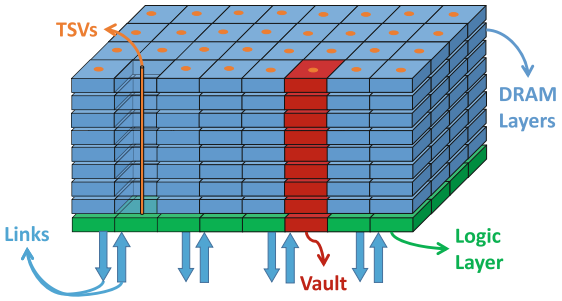

Fig. 1. 3D-stacked memory layout comprising of eight DRAM layers and a base logic layer connected by TSVs and vertically organized in vaults.

processing element type takes advantage in programmability since it inherits the commodity software development tools such as MPI and CUDA. On the other hand, processing elements built based on dedicated circuitry often rely on replication of Functional Unit (FU) elements, which, in turn, achieves high DLP, memory bandwidth, and computational power [6].

2.2 Constraints and Feasibility of 3D-Stacked PIM

Although 3D-stacked PIM is feasible for a broad range of application domains, some design challenges must be faced when considering a 3D memory PIM project. The designers have to deal with power, area and energy constraints for the logic layer to effectively implement the PIM architecture.

According to [1], the power budget related to the logical layer available in the last generation of HMC comprises 11W. However, this constraint can be even smaller if thermal aspects are taken into consideration, reducing it to mere 8.5 W. Regarding area, for an HMC design with a capacity of 8 GB distributed along eight DRAM layers, 16 memory banks and 32 vaults, the area available in the logic layer corresponds to 144 mm^2. Thus, taking into consideration all the physical and technological aspects involved in the logic layer project design, the most suitable PIM processing element type for 3D-stacked memories is that one based on FUs replication [1]. Moreover, FU-centered PIM design with vector operations capabilities can exploit both the entire bandwidth available and provide high DLP, while fitting in the logic layer constraints.

3 Reconfigurable Execution

There are several possibilities to be explored in a PIM architecture design when concerning about energy consumption. Even following the power and area constraints related to 3D logic layer, some aspects intrinsic from DRAM technology cannot be changed. Although PIM reduced the memory walls, the time spent to perform a memory access to a DRAM is still tens of times higher than the time

required to process the data in the FUs. So, exploring such inherent characteristics in PIM designs can lead to an energy consumption reduction.

Recent proposals of PIM architectures rely on multiple homogeneous processing units, and most of them have in common a high number of FUs. For example, we will examine groups of Single Instruction Multiple Data (SIMD) units organized into several Vector Processing Units (VPUs). Thus, sizing the number of FUs to exploit the high internal bandwidth requires an analysis of the area and power costs. Floating point units, for instance, are generally required in basic applications and have a high cost regarding area and power [7]. Energy savings can be achieved by (a) selecting how many units will execute a code portion on compile time (b) dynamically scaling frequency or by (c) reconfiguring the data path to turn on/off FUs when they are not used for a long time.

3.1 Application Classification

The SIMD units with fixed width and frequency cannot satisfy different processing utilization requirements of various applications. Even a single application can have a variable processing power pattern during execution time [8]. PIM based on SIMD units are a natural choice for many data-intensive workloads [9–11], as this combination can benefit both memory-intensive and compute-intensive applications. Some known metrics can identify the compute intensity and cost of memory access, such as:

1. **Memory access distance:** indicates a fraction of cycles where pipeline could be stalled due to demand for load or store instructions. As kernel code portions with poor-temporal locality may cause a miss throughout the cache hierarchies, the major part of such cycles may be spent due to DRAM latency accesses [12].
2. **Arithmetic intensity and cost of instruction:** indicates how many SIMD operations are made in sequence, and the cost in cycles of each instruction. For instance, floating point division takes 20 cycles, while typical bit-wise operations are made in 1 cycle.
3. **Memory parallelism:** the predominance of regular memory access patterns, either by vault-level and bank-level parallelism, reduces the average memory access latency. Thus, the reduced and stable latency can be used to create a pipeline for data stream workloads [13,14].

Either a compiler or a HW mechanism can identify such metrics. As past studies already extracted these characteristics on compile time, a compiler-based tool with hardware information should be suitable for that task. This compiler tool can analyze basic blocks based on the costs of instructions and memory accesses to foresee the arithmetic intensity. To better illustrate this analysis, two basic blocks of common kernels with irregular processing power demands are presented in Listing 1.1. The first basic block (*BB0*) performs a data copy operation from one region of the memory to another, and the second basic block (*BB1*) executes the dot product of two vectors.

In the *BB0*, the whole basic block consists of PIM load and store instructions, which implies that no processing power of SIMD units is needed and all FUs can be turned off. In contrast to the *BB0*, the *BB1* has 14 PIM instructions between the second load instruction and the next memory access in the loop. For instance, the 14 instructions take 52 cycles, or 41.6 ns using the default period (0.8 ns) of the HMC's logic layer, to be completed. As 14 SIMD instructions take 41.6 ns considering 256-byte SIMD units, this represents about 20 GFlops. In order to not hurt the total execution time, the maximum processing power must be kept the same. However, the number of FU per SIMD unit can be significantly reduced without performance degradation, while improving the energy of computation.

In this paper, we rely on a compiler that identifies and offloads instructions to a PIM device as presented in [15]. Then, we test this heuristic to find the optimal point regarding energy-efficiency in a case study architecture. Some alternatives, such as vector size and Dynamic Voltage Frequency Scaling (DVFS) are considered. However, the need for a fine-grain reconfigurability leads to changes in the execution mechanism that will be presented in Sect. 3.3.

Listing 1.1. Example of an assembly code snippet with two different basic blocks.

```
.BB0:
        PIM_256B_LOAD    VPU_0_Reg_0, pimword ptr [rax]
        PIM_256B_LOAD    VPU_0_Reg_1, pimword ptr [rax + 256]
        PIM_256B_STORE   pimword ptr [rbx], VPU_0_Reg_0
        PIM_256B_STORE   pimword ptr [rbx + 256], VPU_0_Reg_1
        add      rbx, 256
        add      rax, 256
        inc      rcx
        cmp      rcx, 16384
        jne      BB0

.BB1:
        PIM_256B_LOAD   VPU_0_Reg_1, pimword ptr [rax]
        PIM_256B_LOAD   VPU_0_Reg_2, pimword ptr [rbx + 4*rcx]
        PIM_256B_VFMUL   VPU_0_Reg_1, VPU_0_Reg_2, VPU_0_Reg_1
        PIM_256B_VFADD   VPU_0_Reg_0, VPU_0_Reg_0, VPU_0_Reg_1
        PIM_256B_VSHUF64x2   VPU_0_Reg_1, VPU_0_Reg_0, VPU_0_Reg_0, 0x3feec
        PIM_256B_VFADD   VPU_0_Reg_0, VPU_0_Reg_0, VPU_0_Reg_1
        PIM_256B_VSHUF64x2   VPU_0_Reg_1, VPU_0_Reg_0, VPU_0_Reg_0, 0x1f4
        PIM_256B_VFADD   VPU_0_Reg_0, VPU_0_Reg_0, VPU_0_Reg_1
        PIM_256B_VSHUF64x2   VPU_0_Reg_1, VPU_0_Reg_0, VPU_0_Reg_0, 0xe
        PIM_256B_VFADD   VPU_0_Reg_0, VPU_0_Reg_0, VPU_0_Reg_1
        PIM_256B_VSHUF64x2   VPU_0_Reg_1, VPU_0_Reg_0, VPU_0_Reg_0, 0x1
        PIM_256B_VFADD   VPU_0_Reg_0, VPU_0_Reg_0, VPU_0_Reg_1
        PIM_256B_PSHUFFLE   VPU_0_Reg_1, VPU_0_Reg_0, 0xffffffffeeffeec
        PIM_256B_VFADD   VPU_0_Reg_0, VPU_0_Reg_0, VPU_0_Reg_1
        PIM_256B_PSHUFFLE   VPU_0_Reg_1, VPU_0_Reg_0, 0xffffffffeedf4e5
        PIM_256B_VFADD   VPU_0_Reg_0, VPU_0_Reg_0, VPU_0_Reg_1
        PIM_256B_STORE   pimword ptr [rcx], VPU_0_Reg_0
        add      rax, 256
        add      rbx, 256
        add      rcx, 256
        inc      r12
        cmp      r12, 16384
        jne      BB1
```

3.2 Vector Size Identification on Compiling Time

Regarding vectorization, traditional open-source compilers like GCC and commercial compilers such as ICC have different approaches to identify vectorization possibilities. Considering the most aggressive compiler flags which enable vectorization, different schedulings for vectorial instructions are related for GCC and ICC assembly codes. The ICC tends to use only vectorial instructions for a given set of elements if their operations can be converted into vectorial versions. When the number of elements is not an exact multiple of the available

vector sizes, ICC issues masked vectorial instructions for those that do not fit entirely in a vector unit. On the other hand, GCC issues as many vector instructions as possible for a specified vector width and scalar instructions for the remaining vector operands. However, none of the state-of-the-art compilers perform an efficient analysis regarding energy consumption in vector operations. Memory-intensive applications could have their processing power reduced without performance degradation by selecting fewer VPUs and turning off the idle ones. Moreover, the number of Load/Store Units impacts the maximum memory bandwidth achievable by an application, which reflects applications where the memory bounds the execution time, rather than the number of processing elements.

3.3 Dynamic Reconfiguration of Execution Mechanism

In a typical vector processor, as the degree of compute-boundness of an application decreases, a significant amount of static energy is lost to keep a great number of FUs in active mode. In applications that are mostly memory-bounded, the full width of a large SIMD unit is used for a few cycles. This fact causes low utilization rate and ineffective energy spendings to keep them in idle mode, mainly when the architecture supports large SIMD units. As the number of active FUs is reduced, the spatial operations of a SIMD instruction can be pipelined in a few number of FUs, and they can be completed with a small penalty of a few more cycles per instruction. As basic blocks may admit a variable increase of the latency of modifying instructions, no performance degradation can be perceived, since the processing latency is masked by the memory access latency in a loop.

A straightforward change in the control unit and data-path of an usual SIMD unit is needed to provide variable width of this unit on the execute stage. Figure 2 presents a possible implementation of a VPU to have a single SIMD instruction pipelined using different factors of two. In this mechanism, the original VPU has 32 active FUs, and Fig. 2A presents 16-FUs setup for compute-bounded basic blocks. For memory-bounded application the SIMD unit can be reconfigured to 2-FUs setup. These different setups provide a fine-grain reconfigurability of SIMD instructions with low cost of reconfiguration. Thus, the PIM device can change this configuration at each basic block or even at each instruction. Further, energy savings are achieved, since static power is avoided by turning off some FUs that would spend more if they were on. The analysis to find the near-optimal setup has also to take into consideration the trade-off between cumulative static energy and execution time.

4 Experimental Setup

In this section, we present the methodology and tools used to evaluate our mechanism.

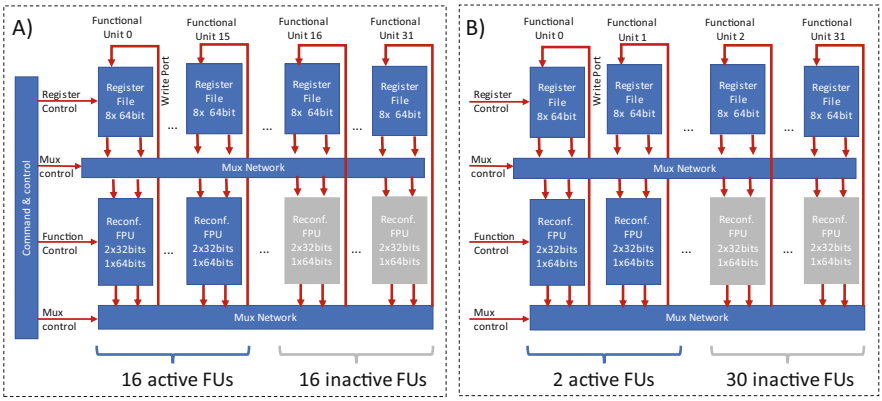

Fig. 2. Example of two reconfiguration setups. Figure A presents half of the total functional units in active mode, thus being able to compute 128 B per cycle. Figure B represent a more constrained processing power capable of executing 8 Bytes per cycle

4.1 The PIM Architecture for a Case Study

The micro-architecture Recon gurable Vector Unit (RVU) [11] was chosen as case study architecture to support our experiments. Accordingly to [1], within a range of recent proposal 3D-stacked PIM architectures, RVU not only fits in the power and area constraints related to HMC logic layer but also is capable of exploring higher memory bandwidth and DLP when compared to others state-of-art PIM architectures. A RVU module comprises a set of 32×8-byte multi-precision FUs, a Finit State Machine (FSM) to control the flow of RVU instructions and a 8×256-byte register file. For the HMC, each *vault* has one RVU module that can operate independently in a parallel fashion. Thus, RVU provides up to 8192-byte FU capacity of vectorial processing and can reach a peak compute power of 2.5 TFLOPS.

The RVU Instruction Set Architecture (ISA) extends the original Intel Advanced Vector Extensions (AVX) keeping compatibility with legacy x86 host instructions allowing a hybrid PIM code style. When the host processor fetches a RVU instruction, it is treated as a store operation and sent to the PIM device to be executed. RVU instructions can deal with operand sizes varying from 4 Bytes to 256 Bytes at once. Additionally, RVU instances can aggregate their execution to deal with bigger instructions ranging from 256 Bytes to 8192 Bytes at once.

4.2 Simulation Method

To experiment and evaluate the proposed techniques, the RVU architecture was implemented for simulation and tests on GEM5 Simulator [16] as presented in [17]. Since RVU extends AVX, and the Intel x86 host processor offloads RVU instructions to HMC, GEM5 was adapted to support both AVX and RVU instructions. For compiling the source code application tests and generating the

binaries, Processing-In-Memory cOmpiler (PRIMO) [15] was used as a support compiler tool. Table 1 summarizes the setup simulated, it comprises an Intel *Skylake* micro-architecture as the host processor and an HMC RVU capable module as main memory.

The energy and power models were obtained by synthesizing the VPU design provided by [1]. This vector unit contains 32×32 bits and 32×64 bits, integer and float-pointing FUs (adders and multipliers), an $8 \times 32 \times 64$ bits register file, and a FSM able to represent a single RVU instance. Supported by Cadence RTL Compiler tool, we extracted area, dynamic and static power for this implementation using 32 nm process technology.

Further, we use a subset of BLAS routines, STREAM benchmark and other miscellaneous kernel applications to represent different kernels behaviors, ranging from mostly memory-bounded to mostly compute-bounded kernels. We varied the number of active FUs from 32 to a single FU, which is given by the #FUn labels in the following charts. Regardless of the data operands present in the benchmarks, a single FU can operate on either 2×32-bit operands or 64-bit operand at a time.

Table 1. Baseline system configuration.

Intel Skylake Microarchitecture
4GHz; AVX-512 Instruction Set Capable; L3 Cache 16MB;
8GB HMC; 4 Memory Channels;
HMC
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;
RVU
1.25GHz; 32 Independent Functional Units; Integer and Floating-Point Capable;
Instructions from 4Bytes to 4096Bytes;
32 Independent Register Bank of 8x256Bytes each;
Latency (cycles): 1-alu, 3-mul. and 20-div. integer units;
Latency (cycles): 5-alu, 5-mul. and 20-div. floating-point units;
Interconnection between vaults: 5 cycles latency;

4.3 Results and Discussion

Figure 3 presents normalized memory bandwidth and processing power achieved by PIM logic to process kernels with different behaviors. Figure 3a depicts a pure streaming behavior where the number of FUs does not impact on the total processing power, neither the average memory bandwidth. As this kernel application is not compute-intensive, the memory bandwidth stands out when the application makes use of the largest load/store instructions available. In contrast to Stream Scale, the Polynomial Solver Equation shows an opposite behavior to

streaming applications, as shown in Fig. 3c. The largest vector widths reach both the highest values of memory bandwidth and processing power. In this case, not only memory bandwidth is required by the application, but also the processing power, which is achieved by the two reconfiguration setups (#FU32 and #FU16). It is possible to notice that the combination of memory- and compute-bound characteristics are found in the Bilinear Interpolation kernel. As shown in Fig. 3b, the discrepancy of bandwidth and FLOPS is only observed on the setups #FU1. One can notice that increasing the vector width also increases the memory bandwidth, thus allowing the use of few FUs to reach the maximum FLOPS.

Figure 4 shows speedup and energy results for the same kernels presented in Fig. 3. To ease the comparison with other designs, the absolute values for each baseline are given in the chart area (Fig. 4). The streaming-like application in Fig. 4a shows that bandwidth limits the speedup. The reconfiguration setup with fewer FUs is enough to consume data and obtain the same performance of the setups with more FUs. To reach a higher performance, more VPUs are required to allow larger load operations. However, this implies that more hardware resources (register file, FSM, and FUs) will be kept in idle mode wasting static power, thus reducing the energy efficiency of those configurations. Similarly, Fig. 4c can reach the highest performance for different reconfiguration

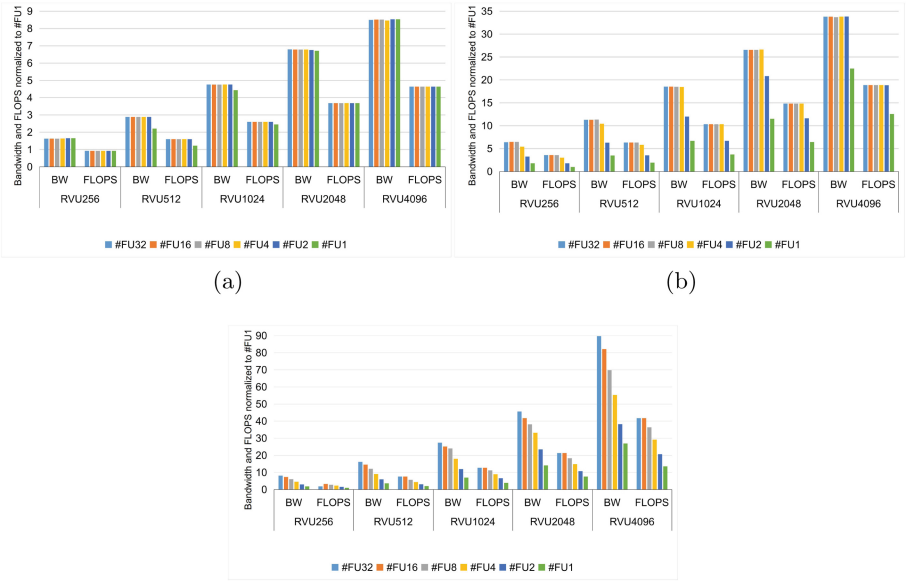

Fig. 3. Total memory bandwidth and processing power for applications with different processing requirements. (a) Stream Scale, (b) Bilinear Interpolation and (a) Polynomial Solver

setups, except for the #FU1. In Fig. 4d, different points can reach the low energy consumption of computation. However, as aforementioned, this application combines memory and compute-bound behavior, which means that the most efficient points will occur when a better compromise between memory bandwidth and processing power. Compute-intensive kernels are profoundly impacted by the number of FUs available in SIMD units, as presented in Fig. 4e. Although the highest performance is reached by using the RVU4096 with setups #FU32 or #FU16, the most energy efficient configuration is achieved by using the setups #FU16 and #FU8.

Despite Fig. 4 has presented different energy consumption and performance points separately, a better metric to show the efficiency of the reconfiguration is the Energy Delay Product (EDP). Figure 5 presents the EDP results of several kernel applications. All columns were obtained by running the largest vector width (RVU4096) and varying the reconfiguration setups. One can notice that memory-bound applications must use fewer FUs to obtain significant energy efficiency. On the other hand, compute-bound applications require higher FLOPS, which is ruled by the number of FUs selected in the reconfiguration.

5 Related Work

There are several works related to exploring energy reduction techniques. The studies mostly associated with PIM architectures, reconfigurable processors and reconfigurable vectorial machines are presented in this section.

Processing-in-Memory: In [18], it is proposed an offload candidate mechanism which can be implemented as a compiler technique. The basic idea is to statically estimate the memory bandwidth savings by whether moving or not blocks of code to be processed near the memory based on dynamic system conditions such as current bandwidth utilization. DRAMA (DRAM-Accelerator) [19] proposes a PIM architecture where the host processor can offload computation and data-intensive operations to Coarse-Grain Recon gurable Arrays (CGRAs) stacked on top of DRAM devices. Similarly, [20] presents Heterogeneous Reconfigurable Logic (HRL), a reconfigurable array for Near-Data Processing (NDP) systems. HRL combines both coarse-grained and fine-grained logic blocks and uses specialized units to support irregular data layouts in analytics workloads effectively. The study represented in [21] reports huge performance speed-up on basic operators of data analytic processing. This PIM architecture achieves significant energy-efficiency by placing SIMD-enabled ARM cores on each HMC *vault*, although it does not support floating point operations and neither presents a technique or optimization to make better use of SIMD units. The work of [10] presents an in-memory resistive design of a general-purpose SIMD co-processor. They claim to allow better scalability and performance compared to a CMOS SIMD processor. However, the main drawback resides on the significantly high power density and low endurance inherent of Resistive Random Access Memories (ReRAM).

Reconfiguration for Energy Reduction: In [22], the authors present a fine grain power-gating technique to cope with future leakage power problem. This

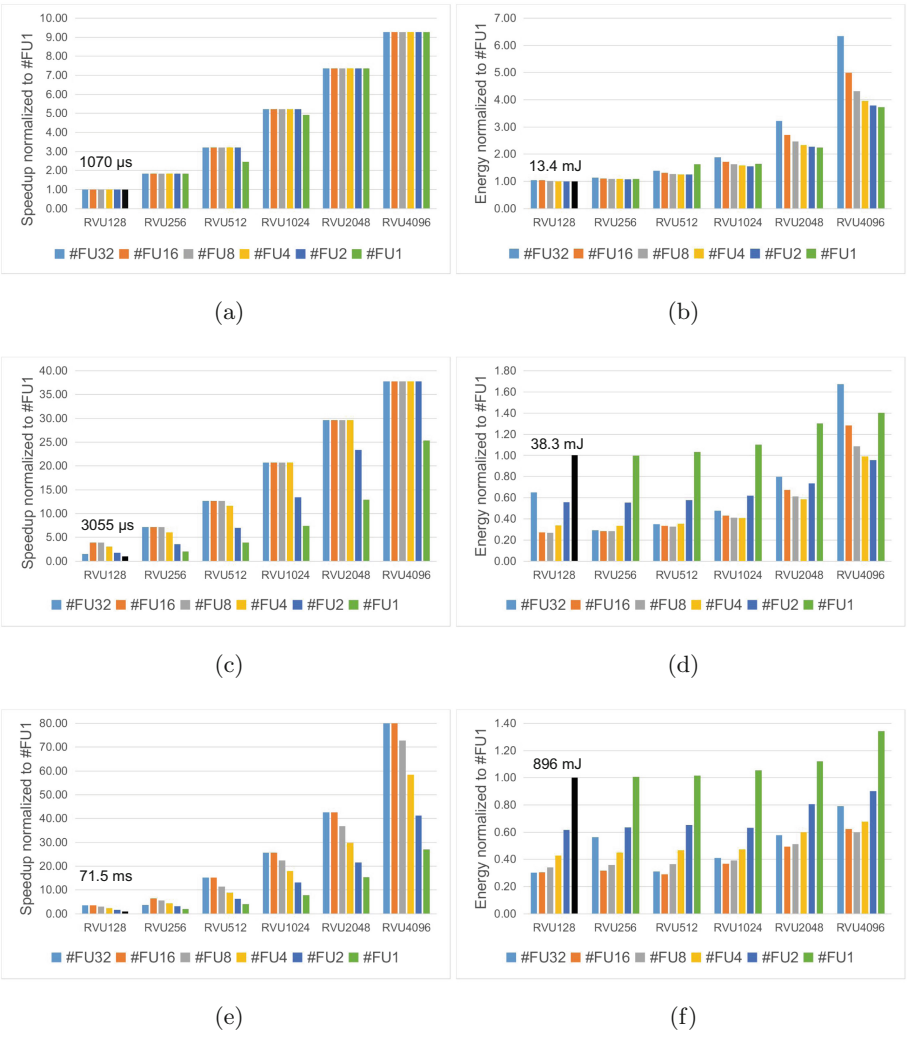

Fig. 4. Speed-up and energy consumption in three applications. (a) and (b) Stream Scale, (c) and (d) Bilinear Interpolation, and (e) and (f) Polynomial Solver Equation

technique can be applied to a CGRA and can reduce up to 48% in real applications. In a different manner, the study of [23] describes a dynamic voltage switching technique to reduce energy dissipation of dynamically reconfigurable processors. This technique dynamically changes the supply voltage of each processing element at the context-by-context basis. However, the energy overhead due to voltage switching hinders the energy reduction, and a mapping optimization was necessary to enable up to 12.5% of total energy savings.

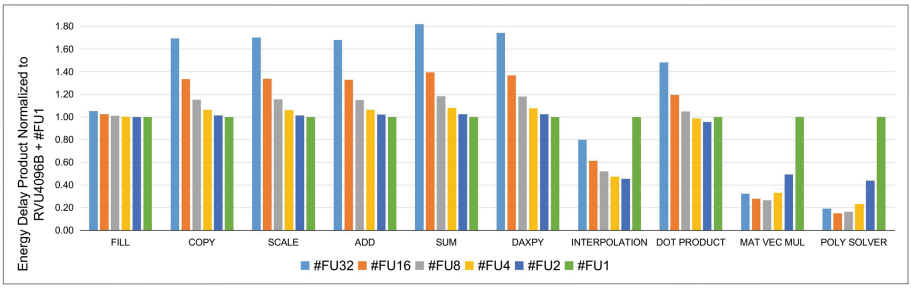

Fig. 5. Energy Delay Product (EDP) results for several application kernels

Reconfigurable Vectors: In [24], the authors propose Softbrain, a reconfigurable vectorial machine for accelerating stream-dataflow applications. Softbrain comprises a control core to generate stream commands, a set of stream-engines to transfer data with memories, and a deeply-pipelined reconfigurable dataflow composed of CGRAs for parallel computation. Regarding regular architectures, [25] proposes an integrated vector scalar mechanism coupled into an ARM micro-architecture core. Their proposed design reuses scalar FUs to provide the execution for vectorial instructions. The main component is a block-based model of implementation that groups vectorial computational operations to execute them in a coordinated manner. ARM Scalable Vector Extension (SVE) defines a SIMD unit able to operate on up to 2048-bit registers, and the SIMD unit defined by the Vector Extension of RISC-V up to 1024-bit registers. However, no physical implementation using the largest size is available yet, and no information regarding power-gating techniques driven by the application's demands on these large registers was found.

6 Conclusions and Future Work

This paper presented a discussion introducing the necessity for the adoption of reconfiguration techniques in vectorial Processing-in-Memory (PIM) architectures to improve energy efficiency. We demonstrated that identifying and taking advantage of the deviations in the compute-intensity to reconfigurable the current PIM architecture can lead to energy savings. To do so, the reconfigurable mechanism must be able to estimate intrinsic applications characteristics. Our simulation results show that, for a set of memory-bounded applications, the number of Functional Units (FUs) on does not interfere in the system performance so that energy savings can be achieved. On the other hand, compute-bounded applications have their memory bandwidth as FLOPS dictated by the biggest number of FUs active. As future works, compiler and hardware techniques for application profiling and PIM reconfiguration will be studied.

Acknowledgment. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and by the Serrapilheira Institute (grant number Serra-1709-16621).

References

1. de Lima, J.P.C., Santos, P.C., Alves, M.A., Beck, A., Carro, L.: Design space exploration for PIM architectures in 3D-stacked memories. In: International Conference on Computing Frontiers, pp. 113–120. ACM (2018)
2. Hu, X., Stow, D., Xie, Y.: Die stacking is happening. IEEE Micro **38**(1), 22–28 (2018)
3. Awan, A.J., Brorsson, M., Vlassov, V., Ayguade, E.: Performance characterization of in-memory data analytics on a modern cloud server. In: 2015 IEEE Fifth International Conference on Big Data and Cloud Computing (BDCloud), pp. 1–8. IEEE (2015)
4. Hybrid Memory Cube Consortium. Hybrid Memory Cube Specification Rev. 2.0 (2013). http://www.hybridmemorycube.org/
5. Lee, D.U., et al.: 25.2 A 1.2 V 8 GB 8-channel 128 GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29 nm process and TSV. In: 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), pp. 432–433, February 2014
6. Zhu, Q., et al.: A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing. In: International 3D Systems Integration Conference (2013)
7. Chen, T., et al.: DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. ACM SIGPLAN Not. **49**(4), 269–284 (2014)
8. Mittal, S.: A survey of techniques for improving energy efficiency in embedded computing systems. arXiv preprint arXiv:1401.0765 (2014)
9. Nair, R., et al.: Active memory cube: a processing-in-memory architecture for exascale systems. IBM J. Res. Dev. **59**(2/3), 17-1 (2015)
10. Morad, A., Yavits, L., Kvatinsky, S., Ginosar, R.: Resistive GP-SIMD processing-in-memory. ACM Trans. Archit. Code Optim. (TACO) **12**(4), 57 (2016)
11. Santos, P.C., Oliveira, G.F., Tome, D.G., Alves, M.A.Z., Almeida, E.C., Carro, L.: Operand size reconfiguration for big data processing in memory. In: 2017 Design, Automation Test in Europe Conference Exhibition (DATE), March 2017
12. Keramidas, G., Petoumenos, P., Kaxiras, S.: Cache replacement based on reuse-distance prediction. In: 25th International Conference on Computer Design, ICCD 2007, pp. 245–250. IEEE (2007)
13. Ding, W., Guttman, D., Kandemir, M.: Compiler support for optimizing memory bank-level parallelism. In: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 571–582. IEEE Computer Society (2014)
14. Sura, Z., et al.: Data access optimization in a processing-in-memory system. In: Proceedings of the 12th ACM International Conference on Computing Frontiers, p. 6. ACM (2015)
15. Ahmed, H., et al.: A compiler for automatic selection of suitable processing-in-memory instructions. In: Design, Automation and Test in Europe Conference and Exhibition (DATE) (2019)
16. Binkert, N., et al.: The gem5 simulator. ACM SIGARCH Comput. Archit. News **39**, 1–7 (2011)

17. Santos, P.C., de Lima, J.P.C., Moura, R.F., Alves, M.A., Beck, A., Carro, L.: Exploring IoT platform with technologically agnostic processing-in-memory framework. In: Proceedings of the Intelligent Embedded Systems Architectures and Applications Workshop. IEEE (2018)
18. Hsieh, K., et al.: Transparent offloading and mapping (TOM): enabling programmer-transparent near-data processing in GPU systems. ACM SIGARCH Comput. Archit. News **44**(3), 204–216 (2016)
19. Farmahini-Farahani, A., Ahn, J., Compton, K., Kim, N.: Drama: an architecture for accelerated processing near memory. Comput. Archit. Lett. **14**(99), 26–29 (2014)
20. Gao, M., Kozyrakis, C.: HRL: efficient and flexible reconfigurable logic for near-data processing. In: 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 126–137. IEEE (2016)
21. Drumond, M., et al.: The mondrian data engine. In: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), pp. 639–651. IEEE (2017)
22. Saito, Y., et al.: Leakage power reduction for coarse grained dynamically reconfigurable processor arrays with fine grained power gating technique. In: International Conference on Engineering and Computer Education (2008)
23. Yamamoto, T., Hironaka, K., Hayakawa, Y., Kimura, M., Amano, H., Usami, K.: Dynamic V_{DD} switching technique and mapping optimization in dynamically reconfigurable processor for efficient energy reduction. In: Koch, A., Krishnamurthy, R., McAllister, J., Woods, R., El-Ghazawi, T. (eds.) ARC 2011. LNCS, vol. 6578, pp. 230–241. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19475-7_24
24. Nowatzki, T., Gangadhar, V., Ardalani, N., Sankaralingam, K.: Stream-dataflow acceleration. In: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), pp. 416–429. IEEE (2017)
25. Stanic, M., et al.: An integrated vector-scalar design on an in-order ARM core. ACM Trans. Archit. Code Optim. (TACO) **14**(2), 17 (2017)