SideDRAM: Integrating SoftSIMD Datapaths near DRAM Banks for Energy-Efficient Variable Precision Computation

RAFAEL MEDINA MORILLAS, Embedded Systems Laboratory (ESL), EPFL, Lausanne, Switzerland
PENGBO YU, Embedded Systems Laboratory (ESL), EPFL, Lausanne, Switzerland
ALEXANDRE LEVISSE, Embedded Systems Laboratory (ESL), EPFL, Lausanne, Switzerland
DWAIPAYAN BISWAS, IMEC, Leuven, Belgium
MARINA ZAPATER, REDS Institute, HEIG-VD, Yverdon-les-Bains, Switzerland
GIOVANNI ANSALONI, Embedded Systems Laboratory (ESL), EPFL, Lausanne, Switzerland
FRANCKY CATTHOOR, Microlab, NTUA, Zografou, Greece and IMEC, Leuven, Belgium
DAVID ATIENZA, Embedded Systems Laboratory (ESL), EPFL, Lausanne, Switzerland

By interfacing computing logic directly to the DRAM banks, bank-level Compute-near-Memory (CnM) architectures promise to mitigate the bottleneck at the memory interconnect. While this computation paradigm heavily reduces the energy requirements for data movement across the system, current solutions fail to co-optimize hardware and software to further increase efficiency. Instead, in this manuscript, we present **SideDRAM**, a co-designed bank-level CnM architecture to enable massively parallel and energy-efficient computations near DRAM. In contrast with past solutions, we support flexible data typing and heterogeneous quantization, relying on the robustness of workloads to employ small bitwidths, and enable a row-wide access to the banks to exploit parallelism and spatial locality. As a result, SideDRAM integrates (1) software-defined SIMD (SoftSIMD) datapaths, supporting low-energy computing with flexible precision, (2) an interface to the banks based on very wide registers (VWRs), enabling asymmetric data access to both utilize the full DRAM bank bandwidth and leverage data locality at the datapath, and (3) a low-overhead distributed control plane, allowing the efficient handling of variable data typing. We benchmark SideDRAM as a near-DRAM solution by analyzing the area, performance, and energy consumption of an HBM2 CnM channel executing heterogeneously quantized machine learning models. The results show that, compared to the state-of-the-art FIMDRAM design, energy improvements of up to 67% are achieved when a DeiT-S inference is executed with a batch size of 16 under the same area constraints, resulting in energy-delay-area product (EDAP)

This work has been partially supported by a joint research grant for ESL-EPFL by IMEC, by the Swiss NSF Edge-Companions project (GA No. 10002812), by the EC H2020 FVLLMONTI project (GA No. 101016776) and by the Swiss State Secretariat for Education, Research, and Innovation (SERI) through the SwissChips research project. This research was partially conducted by ACCESS—AI Chip Center for Emerging Smart Systems, supported by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government.
Authors' Contact Information: Rafael Medina Morillas, Embedded Systems Laboratory (ESL), EPFL, Lausanne, VD, Switzerland; e-mail: rafael.medinamorillas@epfl.ch; Pengbo Yu, Embedded Systems Laboratory (ESL), EPFL, Lausanne, VD, Switzerland; e-mail: pengbo.yu@epfl.ch; Alexandre Levisse, Embedded Systems Laboratory (ESL), EPFL, Lausanne, VD, Switzerland; e-mail: alexandre.levisse@epfl.ch; Dwaipayan Biswas, IMEC, Leuven, Flanders, Belgium; e-mail: dwaipayan.biswas@imec.be; Marina Zapater, REDS Institute, HEIG-VD, Yverdon-les-Bains, VD, Switzerland; e-mail: marina.zapater@heig-vd.ch; Giovanni Ansaloni, Embedded Systems Laboratory (ESL), EPFL, Lausanne, VD, Switzerland; e-mail: giovanni.ansaloni@epfl.ch; Francky Catthoor, Microlab, NTUA, Zografou, Attica, Greece and IMEC, Leuven, Flanders, Belgium; e-mail: catthoor@microlab.ntua.gr; David Atienza, Embedded Systems Laboratory (ESL), EPFL, Lausanne, VD, Switzerland; e-mail: david.atienza@epfl.ch.

ACM 1539-9087/2025/09-ART111
https://doi.org/10.1145/3762641

savings that reach 83%. When comparing to a massively parallel mixed-signal CnM solution, SideDRAM consistently obtains similar performance and better energy efficiency results (geomean of 15× improvement across workloads) at a lower area overhead.

CCS Concepts: • **Computer systems organization** → **Parallel architectures**; • **Computing methodologies** → **Neural networks**; • **Hardware** → **Hardware accelerators;**

Additional Key Words and Phrases: Compute-near-memory, DRAM, energy-efficient computing, heterogeneous quantization, distributed control plane, processing-in-memory

ACM Reference Format:
Rafael Medina Morillas, Pengbo Yu, Alexandre Levisse, Dwaipayan Biswas, Marina Zapater, Giovanni Ansaloni, Francky Catthoor, and David Atienza. 2025. SideDRAM: Integrating SoftSIMD Datapaths near DRAM Banks for Energy-Efficient Variable Precision Computation. *ACM Trans. Embedd. Comput. Syst.* 24, 5s, Article 111 (September 2025), 24 pages. https://doi.org/10.1145/3762641

1 Introduction

The growth in complexity and size of modern artificial intelligence and other data-intensive applications [49] implies an increasingly large number of data transfers between compute logic and memory. This behavior clashes with the memory wall problem [50], as the performance disparity between processing and memories has only broadened in the last few decades [13]. Not only do the interconnects not sustain the required bandwidth, but also significantly contribute to system energy consumption [22, 33].

As traditional Von Neumann architectures struggle to address these limitations, novel **Compute-in-Memory** (**CiM**) and **Compute-near-Memory** (**CnM**) solutions have emerged to satisfy the bandwidth and energy efficiency needs by bringing computation and storage together [24]. CiM approaches leverage memory structures to perform computations among memory rows [15, 31, 36]. While this method allows row-wide parallelism and low energy use, it incurs the steep costs of modifying the IPs of the memory array. Instead, CnM architectures interface processing capabilities in close proximity to the memory while avoiding its modification, enabling high-bandwidth access to perform energy-efficient, parallel computations [4, 7, 8, 10, 17, 23, 25, 28–30, 33, 34, 38, 42, 43, 45].

Although CnM can be implemented at any level of the memory hierarchy [44], interfacing **processing units** (**PUs**) directly to the DRAM banks offers a very high-bandwidth access to the data. This approach, herein referred to as bank-level CnM (shown in Figure 1(a)), minimizes the energy employed for data movement and preserves the optimized subarray design. Furthermore, access to the banks can be parallelized within a channel to increase attainable throughput [17, 28, 29].

However, the bank-level architectures presented to date [7, 10, 17, 28, 29] have not fully taken advantage of the algorithmic quantization opportunities brought about by the machine learning era, often being limited to FP16 and INT8 computation. Thus, computation efficiency and quantization compatibility are restricted. Moreover, they comprise resource-hungry multipliers [34] or mixed-signal ALUs [42], resulting in high area and energy overheads that are particularly critical under the stringent constraints at the bank periphery. These design choices also impact the achievable parallelism. Since increasing the bandwidth of the interface to memory requires equivalent growth in area for computation, the PUs need to limit the width of data accesses to column granularity [7, 10, 17, 28, 29, 34], failing to leverage the full parallelism potential at the bank periphery.

To address these challenges, we draw upon low-power microarchitectural techniques [6] that strive for high parallelism, flexible data precision, and high locality under tightly constrained area and power budgets. Specifically, we identify **software-defined SIMD** (**SoftSIMD**) [27, 52] as a promising CnM strategy to support massive DRAM parallelism with variable precision at minimal

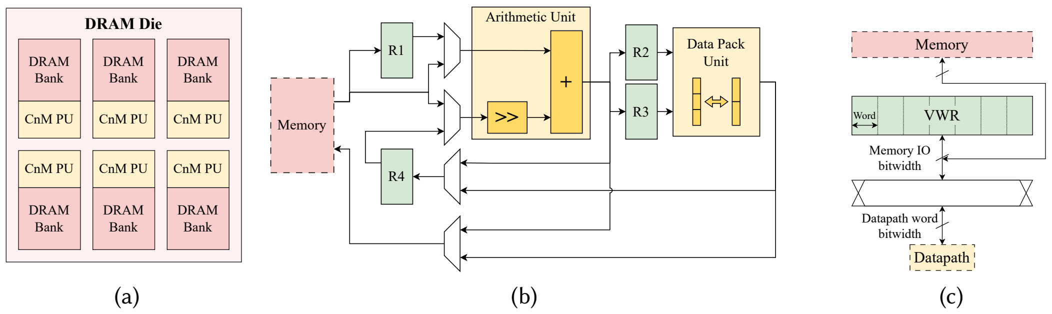

Fig. 1. Diagrams of (a) a CnM DRAM, (b) a SoftSIMD datapath, and (c) a VWRs.

area and energy overheads. In this context, our approach in this article is threefold. First, we implement SoftSIMD datapaths [27, 52] to enable instruction-controlled flexible operand precision at runtime, improving utilization, area, and energy efficiency of the PU. Consequently, low overhead and highly scalable SoftSIMD datapaths can exploit the massive parallelism at the DRAM bank periphery. Second, to leverage the full memory bandwidth at a high energy efficiency, we integrate **very wide registers** (**VWRs**) [9, 40]. These offer a single-port alternative to multi-port register files through an asymmetric interface, with a wide access towards memory and a narrower one towards the datapath, which simplifies decoding logic and allows to profit from spatial locality. Profiting from these characteristics, we interface row-wide VWRs directly to the bank row buffers, and provide the SoftSIMD logic with datapath-wide words. Thirdly, to govern execution of the highly parallel SoftSIMD datapaths and VWRs at low area and energy overheads, we design a modular control plane based on **distributed loop buffers** (**DLBs**) [6, 20]. This approach improves scalability and energy efficiency by placing the control logic close to the corresponding datapath **functional units** (**FUs**).

Hence, we introduce **SideDRAM** (SoftSIMD Datapaths near DRAM), a novel bank-level CnM domain-specific architecture that integrates low overhead, very scalable SoftSIMD datapaths close to the banks, interfaced through energy-efficient row-wide VWRs, and governed by a low-overhead control plane based on DLBs. We assess the area, performance, and energy consumption of different configurations of the architecture when executing matrix multiplication kernels and heterogeneously quantized machine learning workloads. This exploration allows us to demonstrate the benefits of SoftSIMD as a CnM strategy, showing that SideDRAM performs highly energy-efficient variable precision computations by profiting from the quantization robustness of workloads and from the full IO bandwidth next to the DRAM banks. In summary, the contributions of this article are the following:

— We present SideDRAM, a novel CnM architecture that co-optimizes flexible data typing and the high bandwidth and locality near the DRAM bank to support massively parallel, energy-efficient variable precision processing. Consequently, the design can effectively host state-of-the-art heterogeneous quantized machine learning models.
— We show how the SideDRAM design can seamlessly integrate to DRAM thanks to a scalable datapath, a flexible asymmetric interface and a minimalist distributed control scheme, leading to area- and energy-efficient computing with arbitrary bitwidths at the constrained bank periphery.
— We analyze the area, performance, and energy results of various configurations of SideDRAM executing heterogeneously quantized ML applications. Compared to state-of-the-art bank-level CnM solutions, we demonstrate that SideDRAM allows to reduce energy consumption

across workloads, reaching savings of up to 67% when executing DeiT-S in 16 batches under the same area constraints as the FIMDRAM architecture. SideDRAM configurations also outperform a recent mixed-signal CnM solution exploiting row-wide parallelism, improving energy efficiency by an average of 15× across workloads while incurring less than 12% of the area overhead.

2 Background

2.1 Bank-level Compute-near-Memory

CnM designs placing PUs close to the DRAM have proposed different levels of integration, from interfacing compute logic to a whole channel or die [8, 23, 33, 38, 43], to attaching it to the memory subarrays [4, 30, 45]. This choice defines the tradeoff between the design effort (simpler at high hierarchy levels) and the energy efficiency and parallelism (better at lower levels) [24, 44]. Among these options, bank-level CnM solutions strike an attractive balance [7, 10, 17, 25, 28, 29, 34, 42]. In fact, interfacing CnM PUs to the banks, as shown in Figure 1(a), benefits both from the high bandwidth available at the bank IO and from energy savings due to shorter data movements, while it avoids the high design costs of redesigning the bank itself. Furthermore, computations can be performed simultaneously across the banks of a channel, again increasing bandwidth.

2.2 SoftSIMD

Single Instruction Multiple Data (**SIMD**) is a well-established paradigm that exploits data-level parallelism to enhance computational efficiency [19]. However, traditional SIMD implementations are restricted to a fixed set of operand precisions, limiting their applicability to mixed-precision workloads. Instead, **Software-defined SIMD** (**SoftSIMD**) [27, 52] enables the support of flexible data quantization with minimal hardware overhead.

To support this behavior, the SoftSIMD datapath can be partitioned into subwords of arbitrary size at runtime, employing bitmasks as a software mechanism to separate subwords and prevent overflow. When adding two n-bit fixed-point SoftSIMD operands, only the lower $(n-1)$ bits encode valid data, preventing the potential generation of a $(n+1)$-bit result. Meanwhile, the **most significant bit** (**MSB**) is used as a guardbit (set to '0' during addition) to avoid carry propagation to the next subword. This behavior is illustrated in Figure 2(a). Similarly, subtraction operation is realized by setting the guardbits to '1', providing a carry-in of '1' and enabling the two's complement computation $A - B = A + \bar{B} + 1$. Shift operation is also supported by performing sign extension for the guardbits and regular shifts for the non-guardbits, as shown in Figure 2(b).

As shown in Figure 2(c), the SoftSIMD datapath also supports multiplication operations with shifts and additions/subtractions. In the example in the figure, both the multiplicand and multiplier operands are encoded in Q1.X format (i.e., 1-bit for integer and X-bit for fraction), yielding values in the (-1, 1) range. The multiplication operation is decomposed into an iterative sequence of arithmetic right shifts and additions/subtractions, processing the multiplier operand from LSB to MSB. To reduce the amount of required iteration cycles, the multiplier operand is encoded with **Canonical Signed Digit** (**CSD**) [2], which represents data with three digits, '0', '1', and '−' (-1). This redundancy is exploited to reduce the number of non-zero digits for a n-bit multiplier operand to an average of $n/3$ [37]. Consequently, the number of additions or subtractions required for multiplication is greatly decreased, boosting performance and energy efficiency. The described method allows to support **multiply-accumulate** (**MAC**) operations on the SoftSIMD datapath, which in turn can be composed to support more complex operators such as dot product, **general matrix multiplication** (**GEMM**), and convolution, as summarized in Figure 2(d).

An example of a SoftSIMD datapath enabling the above-mentioned functionality was introduced in [52]. As shown in Figure 1(b), it comprises an **arithmetic unit** (**AU**), a **data pack unit** (**DPU**),

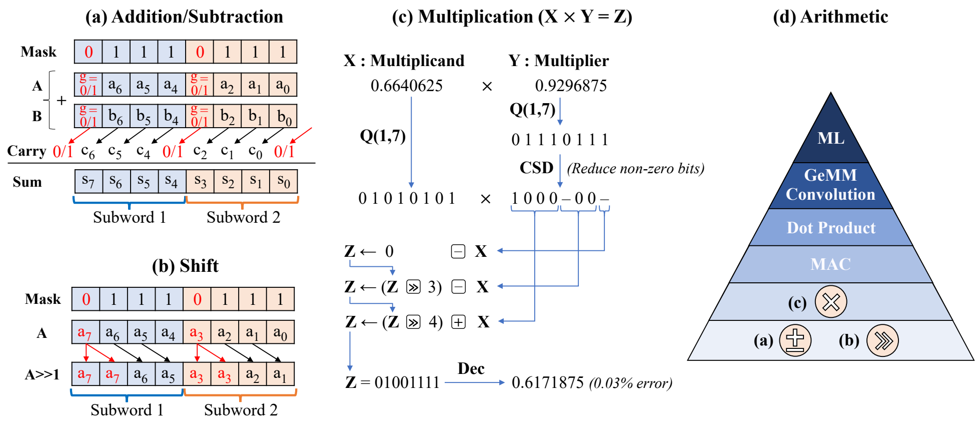

Fig. 2. Arithmetic operations on the SoftSIMD datapath: (a) addition/subtraction, (b) shift, (c) multiplication, and (d) more complex operations composed by the former three, enabling support of ML kernels. Multiplication is implemented as an iterative sequence of shift-and-add operations, with the multiplier operand encoded in CSD form to minimize cycle count.

and four registers (R1–4). The AU supports vector addition, subtraction, and shifting with flexible SIMD modes configured at runtime, thus, also allowing to perform vector by scalar multiplication. Data inputs are fetched either from memory or from the local registers R1 and R4, which helps to avoid frequent memory reads in shift-add or accumulation operations. The output of the stage can be written back to memory, stored in R4 for local reuse, or sent to R2 or R3 to be repacked in the DPU.

The DPU allows to repack subwords from R2 and R3, bridging across different precision levels. It supports two different conversion modes. First, the DPU can increase or decrease operand precision to support accumulation and truncation operations. Thanks to the dynamic precision management, expensive accumulators are no longer required. Second, shuffling of subwords is supported for local data storage, maintaining precision. The output of the DPU can be written back to memory, or stored in R4 for future reuse.

2.3 Very Wide Register

VWRs [6, 40] were introduced as an energy-efficient alternative to traditional multi-port register files. As shown in Figure 1(c), they implement a single register-wide port, which avoids the energy overhead incurred by decoders and multiple ports. In addition, they present an asymmetric interface. The VWR connects to memory through a bus as wide as its IO, leveraging the full memory bandwidth to transfer multiple datapath words simultaneously, reducing the overall energy per word access. When communicating with the datapath, the data is instead multiplexed to interface at the word width, profiting from data locality. The design of the VWR makes it inherently scalable thanks to the simplicity of the decoding needed and the single IO port. In addition, place and route is streamlined as the memory IO and the VWR can be aligned to minimized wire length [6, 9, 40]. This property makes VWRs specially convenient in the DRAM periphery, where the area restrictions and reduced number of metal layers make difficult the implementation of more complex register files. We draw on these characteristics to interface VWRs to the DRAM row buffers, aligning with the DRAM bank organization, as detailed in Section 4.1.

2.4 Heterogeneous Quantization

Quantization [12, 32, 35] is a widely adopted method that converts floating-point numbers to fixed-point representations, thereby reducing the memory footprint and computational overhead while maintaining accuracy. This technique is especially crucial for edge devices with limited resources and energy budgets. The most common implementation, uniform quantization [3, 35], compresses weights and/or activations into fixed integer formats. However, this approach overlooks variations in layer robustness, as certain layers can tolerate more aggressive precision reduction without significant performance loss. Alternatively, heterogeneous quantization [14, 39, 46, 52] assigns different precision levels for weights and/or activations across layers. This allows to use low precision in less sensitive layers, while maintaining high precision in critical ones. However, hardware support for heterogeneous quantization is not widespread, as most architectures accommodate only a narrow range of operand formats. In contrast, our integrated SoftSIMD PUs address this limitation by allowing flexible operand bitwidths at runtime, thus, supporting heterogeneously quantized workloads.

3 Related Work

While recent bank-level CnM works show a wide variety of designs, implementing general purpose [10], domain-specific [25, 29], and application-specific [7, 17, 28] architectures, most of them access the DRAM bank through the column decoder, forfeiting the full bandwidth available at the periphery. Consequently, many computation cycles are devoted to data movement between memory and large CnM local register files [25, 29]. In addition, CnM parallelism and energy efficiency are further limited by the implementation of several multipliers per PU, incurring high area and energy costs at the stringently constrained bank periphery. This overhead is further increased by including per-PU control logic and registers [7, 10, 25, 29]. Although Compute-in-DRAM alternatives [15, 31, 36] aim to address these limitations by leveraging the DRAM dynamics to process using entire rows, they require modifications in the memory array structures that entail very high non-recurring engineering costs.

In this work, we instead propose to interface the PUs to the global row buffer to take advantage of the available high bandwidth and high locality. To enable this, we employ highly scalable and regular VWRs, SoftSIMD datapaths, and a DLB-based control scheme. By connecting VWRs directly to the sense amplifiers, we allow easy asymmetric access that leverages locality and reduces wiring overhead. In turn, the multiplier-less SoftSIMD datapaths support flexible precision at runtime to maximize attainable parallelism at the PU while keeping energy costs low. Finally, implementing a control plane based on DLBs allows to share the majority of the control logic among several PUs in a channel, decreasing the necessary area and energy at a reduced interconnection cost.

Our approach is related to the recent TULIP-DRAM [42, 48], which proposes a similar integration at the row buffer, employing configurable neurons implemented as mixed-signal circuits based on threshold logic to exploit DRAM parallelism. However, integrating analog solutions at the DRAM periphery remains a challenge entailing high design efforts. The targeted parallelism also causes significant area and energy overheads, as explored in Sections 6 and 7. Moreover, as each configurable neuron can only process a few bits at a time, the computations in TULIP-DRAM incur long delays and require extensive data replication to increase throughput. In contrast, our proposed SideDRAM architecture exclusively employs digital circuits, facilitating its implementation at the bank IO, and achieves competitive performance with higher area and energy efficiency without relying on data replication to enhance parallelism.

4 SideDRAM Architecture

The SideDRAM architecture, presented in Figure 3, comprises three main elements: a series of very wide registers (VWRs, Figure 3(b)) tightly integrated with each DRAM bank IO, a SoftSIMD datapath (Figure 3(c)), and a control plane (Figure 3(d)), which in turn is composed of a global control plane that can govern the simultaneous execution of various datapaths, and local control logic

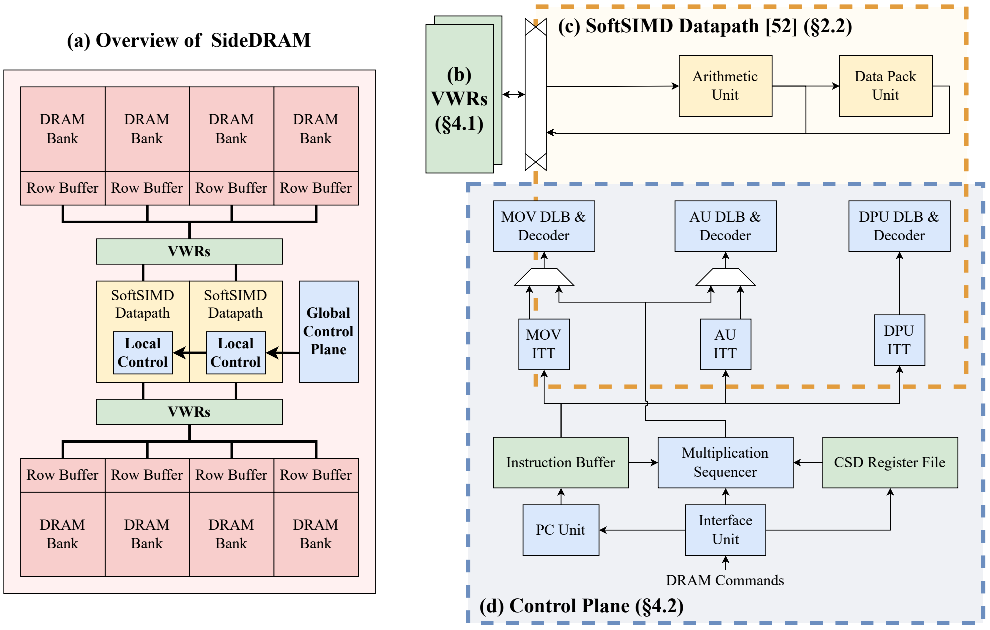

Fig. 3. Diagrams illustrating the integration of SideDRAM at the DRAM bank periphery. The left figure (a) shows an overview of the SideDRAM architecture. The right figure details how the different elements are interconnected: the VWRs (b) interface the DRAM banks, accessing the entire row buffer width, to the SoftSIMD datapaths (c). The data movement and the execution of the datapaths are governed by a shared distributed control plane (d).

decoding the instructions at each datapath. The combination of these elements enables the area- and energy-efficient support of linear algebra kernels widely used in data-intensive applications [1, 25].

Integrating SoftSIMD datapaths next to the DRAM periphery allows to leverage the available massive parallelism, achieving high resource utilization. Moreover, it enables to balance computation precision and workload robustness to enhance energy efficiency without compromising performance, drawing on the heterogeneously quantization of workloads. As detailed in Section 2.2, the datapath includes two pipeline stages (the arithmetic unit, AU, and the data pack unit, DPU) and four local registers (R1–4). It also has access to two VWRs, which allow reading and writing of datapath-wide words. The elements of the SoftSIMD datapath are clock-gated to comply with low-power requirements.

The modularity of the datapaths, VWRs, and control plane allows assembling different architectural configurations according to varying hardware and software constraints, as explored in Section 6. These designs place one or several SoftSIMD datapaths at the DRAM bank periphery, as depicted in Figure 3(a), sharing the same global control plane. They are interfaced with the banks through two VWRs. In turn, each of these VWRs can read/write from/to one or more DRAM banks, enabling bidirectional communication one row at a time. The rest of this section will describe in detail each of the SideDRAM architectural elements and how they are integrated into the system.

4.1 VWR Interface between Datapath and DRAM

VWRs serve as the intermediary between the SoftSIMD datapath and the DRAM banks. Their width matches that of the DRAM row and, to profit from the full bank IO bandwidth, they send and

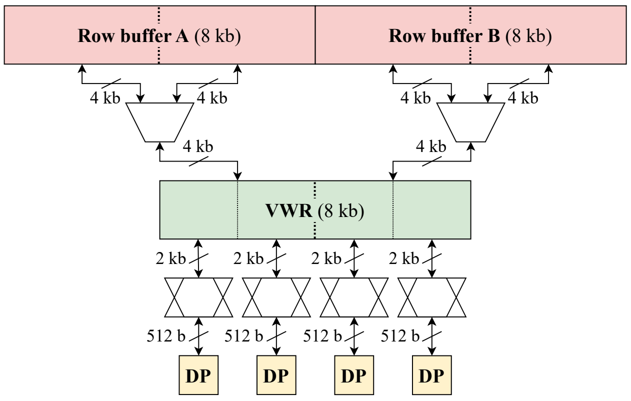

Fig. 4. Example of a VWR serving as interface between two row buffers and four datapaths. The width of the row buffers and the VWR is 8 kb, while the width of the datapaths is 512 b. Each half of the VWR is connected to one row buffer, and the slice of the row buffer written is selected according to the DRAM bank address bits. On the SoftSIMD side, each datapath can access a 512-bit word multiplexed from a 2-kbit slice of the VWR.

receive data directly to and from the row buffers. An example of the VWR interface is shown in Figure 4. The VWRs can connect to one or several banks. In the latter case, each of the interfaced banks sends or receives a slice of the VWR, i.e., if a row-wide VWR is connected to two banks, it reads or writes a slice as wide as half a row to each bank in parallel. To access the data in the VWR, the datapath can read and write datapath-wide words once per cycle. This asymmetric interface improves the energy efficiency of the data movements, as one row read from DRAM to VWR is enough for the execution of numerous SoftSIMD instructions, leveraging data locality. In case a VWR containing W words is shared among N datapaths, each of them can access W/N words.

In addition to profiting from data locality and high bandwidth access, employing VWRs to interface the SoftSIMD datapaths and the DRAM banks facilitates compliance with the implementation constraints at the periphery. As discussed in Section 2.3, VWRs are scalable thanks to their single IO port and simple decoding logic. Furthermore, VWRs are aligned with the interfaced row buffers to ease place and route.

4.2 Control Plane

The control plane of SideDRAM governs both the execution of the SoftSIMD datapaths and their interface with the DRAM banks through the VWRs. Its design presents high area and energy efficiency, while flexibly supporting arbitrary datapath widths and data types. Its modular structure is based on **distributed loop buffers** (**DLBs**). These were introduced as an alternative to traditional VLIW buffers [6, 20] to avoid their limitations. Namely, their need for wide wordlines to hold all the micro-instructions (μ-instructions) for the FUs, and their centralized decoding approach requiring long wires to deliver the control signals to the FUs. Instead, DLBs partition the instruction buffers and cluster the divisions according to functionality. As shown in Figure 5, this method allows the placement of local buffers and decoders physically close to the FUs they control, reducing wiring and energy overhead. Moreover, such an approach is enhanced with the use of **Index Translation Tables** (**ITTs**, in Figure 5) that allow to employ a single general index to address the correct μ-instructions in each DLB. For this purpose, each row in the ITT contains two fields signaling if the corresponding FU is active, and with which index the DLB should be addressed. Consequently, DLBs do not contain empty addresses, and unnecessary decoding of instructions is avoided.

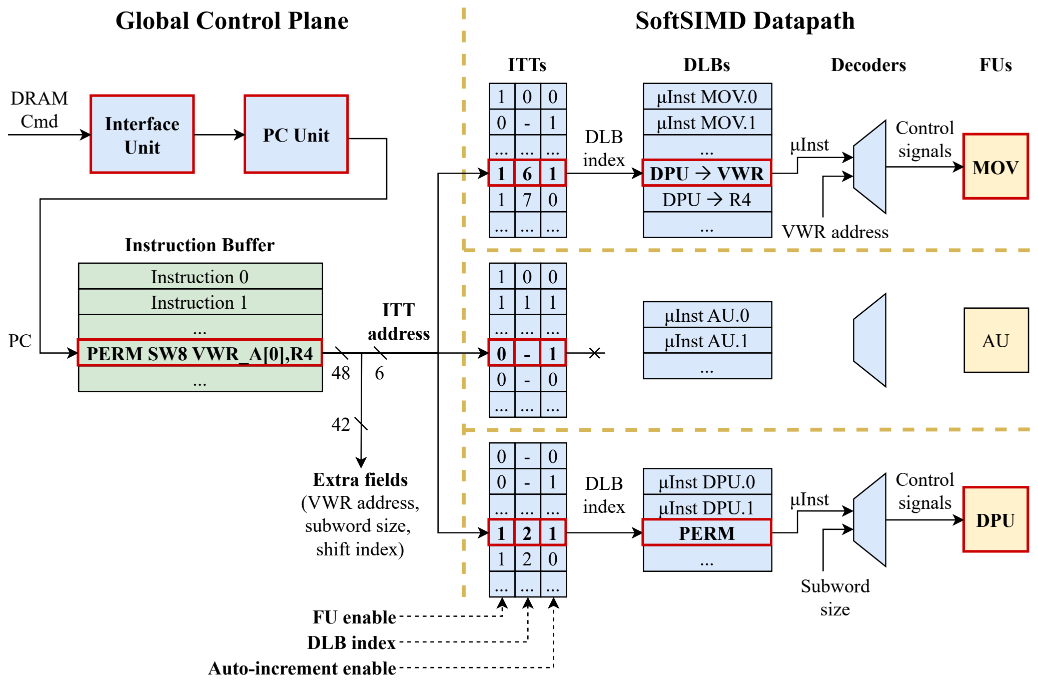

Fig. 5. Diagram depicting the translation process from the instructions in the global control plane to the control signals for each of the FUs in the SoftSIMD datapath, exemplified with a PERM instruction writing its result to VWR_A and R4. The elements active during the translation are highlighted in red.

As a result of integrating DLBs, the control plane (Figure 3(d)) is composed of two main blocks: the global control plane, whose output can be distributed among several datapaths, and the local control logic, which is placed in close proximity to the datapath logic and interprets the information received from the global control plane. In turn, these two blocks comprise different logic elements that allow the control of the datapaths, which can be seen in Figure 3(d). The local control logic at each datapaths consists of the DLBs and local decoders used to generate the control signals to guide SoftSIMD execution, together with the ITTs. Meanwhile, the global control plane includes the instruction buffer, the interface unit, the PC unit, and the multiplication sequencer with the CSD register file (both needed for SoftSIMD multiplication through shifts and additions). This structure shortens the distance between the datapath elements and the ITTs, DLBs, and decoders, facilitating scalability and decreasing energy cost of the control logic. Moreover, the largest control elements (i.e., the instruction buffer, the CSD RF, and the multiplication sequencer described below) are located in the global control plane, reducing area requirements. While we only consider a single control plane in the explorations in Sections 6 and 7, it can also be replicated to assist scalability when extensively increasing the number of VWRs and datapaths in the architecture. The rest of the section presents in detail the elements of the control plane and the instruction translation process.

4.2.1 Interface Unit. Overall execution of the SideDRAM architecture is governed by the DRAM command and address pair. By interpreting them as described in Section 4.3, the interface unit can program the instruction buffer, CSD RF, and hardware loop registers, as well as trigger the decoding of the next instruction pointed by the program counter.

Table 1. ISA of the SideDRAM Architecture

NOP CLKS	Multi-cycle no-operation
EXIT	End of CnM execution
RLB VWR_SEL	Read from the row buffer to a VWR
WLB VWR_SEL	Write from a VWR to the row buffer
VMV VWR_SEL IDX	Move a word from a VWR to R1
RMV VWR_SEL IDX	Move the word in R4 to a VWR
PACK SW_CHANGE OFFSET DST_SEL	Repack the contents of R2 and R3, changing subword size
PERM SW_LEN OFFSET DST_SEL	Permute the contents of R2 and R3
SHIFT N_SHFT SW_LEN SRC_SEL DST_SEL	Arithmetic right shift of a word
ADD SW_LEN SRC_SEL DST_SEL	Addition of two words
SUB SW_LEN SRC_SEL DST_SEL	Subtraction of two words
MUL SW_LEN CSD_IDX DST_SEL	Multiplication of a word by a scalar

4.2.2 PC Unit. This block generates the program counter that indexes the instruction buffer. It also supports hardware loops, storing in dedicated registers the start and end addresses, the total number of iterations, and the index of the current iteration.

4.2.3 Instruction Buffer. The instructions describing the high level behavior of the connected SoftSIMD datapaths and VWRs are stored here. As shown in Table 1, the resulting ISA can command control (NOP, EXIT), data movement (RLB, WLB, VMV, RMV), subword manipulation (PACK, PERM), and arithmetic operations (SHIFT, ADD, SUB, MUL). Each 48-bit instruction contains two parts: the address to access the ITTs, and fields common to all the FUs that will be sent to the corresponding decoders. These fields support addressing a word within the VWRs, specifying the subword size and size change, and defining the number of right shifts at the AU.

4.2.4 Index Translation Tables. The ITTs generate the DLBs indices according to the address received from the instruction buffer. For this purpose, each row in the table includes three fields, as seen in Figure 5. The first one indicates whether the associated FU is active, avoiding unnecessary decoding energy. Then, the second field contains the translated index to access the corresponding DLB. Since some instructions imply the sequential execution of DLB μ-instructions in several cycles, the ITT address is auto-incremented, starting from the one received from the instruction buffer. Thus, the third field signals whether to access the next ITT address after the current one or to wait for the next instruction.

4.2.5 Distributed Loop Buffer and Decoders. Each of the DLBs contains the μ-instructions that describe the possible behavior of their corresponding FU: data movement unit (MOV), AU, and DPU. The μ-instructions can be reused by several instructions, e.g., the μ-instruction describing addition is involved in both ADD and MUL operations. The DLBs and decoders are placed closed to the FU they govern, generating the relevant control signals according to the selected μ-instruction and the common fields provided by the instruction buffer.

4.2.6 Translation of Instructions into Control Signals. When the appropriate DRAM commands arrive to the SideDRAM architecture, they prompt the translation from instructions to the control signals for the SoftSIMD FUs. An example of this process is showcased in Figure 5. Here, the interface unit interprets the arriving DRAM command, causing the decoding of the next instruction. Consequently, it triggers the PC unit to generate the next PC selecting the instruction. In the example, a PERM instruction is selected, which permutes the subwords contained in registers R2 and R3, and writes the resulting words in VWR_A[0] and R4. The six most significant bits of the instruction are used to index the corresponding rows of the three ITTs. Each of these rows comprises three fields: (1) FU Enable, determining if the corresponding FU should be active for the current instruction, (2) DLB index, specifying the index of the DLB containing the μ-instruction to be executed in the FU, and (3) Auto-increment enable, which allows to support multi-cycle instruction by signaling whether the ITT index should be incremented. According to these, the `PERM` instruction in Figure 5 employs the MOV and DPU FUs, avoiding any further decoding for the unused AU. The ITTs also provide the index of the MOV and DPU μ-instructions, describing a movement from the DPU to the VWRs and a permutation, respectively. The auto-increment enable bit of the ITT row also indicates that, to complete the current instruction, the next row should also be accessed. Following this, the indexed μ-instructions are decoded together with the relevant fields from the instruction to generate the control signals governing execution. In the example, the extra fields specify VWR address (`VWR_A[0]`) and subword size (`SW8`).

4.2.7 Multiplication Sequencer and CSD Register File. SoftSIMD multiplication is performed through a series of shifts and additions/subtractions, as detailed in Section 2.2. To support this process, additional control logic is needed to generate the corresponding sequence of μ-instructions. First, the control plane includes a CSD RF that stores the scalar multipliers needed for multiplication. As mentioned in Section 2.2, these operands are represented in CSD form to reduce the amount of non-zero digits in the encoded number, which results in less additions and subtractions needed for computation. During execution, when a `MUL` instruction is reached at the instruction buffer, the multiplication sequencer is activated, receiving the corresponding CSD multiplier from the register file. Then, employing a finite state machine, it progressively reads the CSD word from the least to the most significant bit to generate the necessary indices for the MOV and AU DLBs. In this manner, and following the strategy in Figure 2(c), the sequencer guides the iterative shifts and additions/subtractions according to the distance between non-zero digits and their sign ('1' or '-').

4.3 System Integration

SideDRAM is integrated into the system main memory, complying with the requirements of the JEDEC standards. Similar to previous works [28, 29, 34], its execution is governed via DRAM commands to specific addresses. By writing to a memory-mapped address, SideDRAM can toggle between memory mode, in which it behaves as a normal (non-compute) memory, and CnM mode, allowing to write programs to the control plane and to execute them.

In CnM mode, the instruction buffer and the CSD register file can be programmed via memory-mapped addresses, storing the instructions and the CSD-coded multipliers, respectively. Once the program is written, the DRAM command and address pairs to the channel trigger the decoding of the instructions in the buffers, prompting the translation into the DLB μ-instructions. Consequently, the program is executed in lockstep across all the datapaths in the channel, executing the same operation on different vectors. If the instructions imply data movement between the DRAM and the VWRs, the DRAM address specifies which row (and which slice, if several banks are interfaced to the same VWR) is being read or written. To ensure correct access sequences we assume a modified memory controller behavior during CnM mode, as in the literature [34, 44, 51], to avoid squashing or reordering DRAM commands.

5 Experimental Setup

5.1 Executing GEMMs on SideDRAM

SideDRAM supports the execution of linear algebra kernels with multiple levels of bit quantization. In order to compare the architecture with state-of-the-art designs, we focus on GEMM as test vehicle, as they represent the backbone of artificial intelligence workloads, which greatly benefit

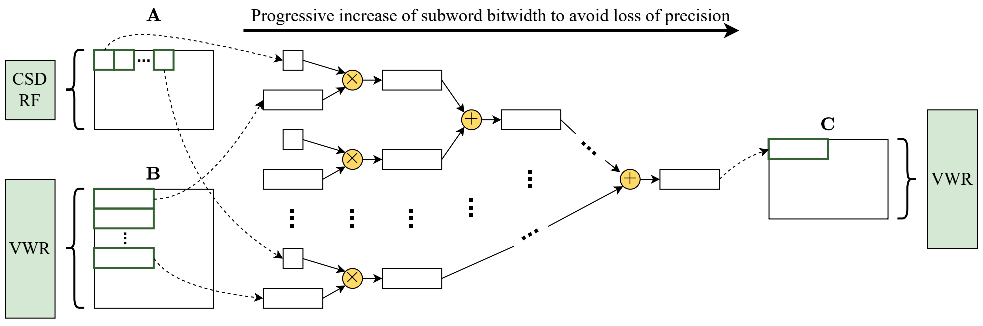
A
Progressive increase of subword bitwidth to avoid loss of precision
CSD RF
B
VWR
C
VWR

Fig. 6. Mapping of a GEMM ($A_{m\times n} \times B_{n\times p} = C_{m\times p}$) to the SideDRAM architecture. The elements of A are encoded in the CSD representation and held in the CSD RF, while the elements of B are stored in the VWR with a given initial subword width. During execution, the bitwidth progressively increases to avoid accumulation overflow. The results of the GEMM (C) are also held in the VWR.

from heterogeneous quantization opportunities [1, 25, 52]. In Section 5.3, these are composed to assess the behavior of entire artificial intelligence workloads.

An instruction generator is used to translate from assembly to the sequence of DRAM commands required to execute GEMMs of arbitrary size and quantization, similarly to [34]. GEMM kernels, computing $A_{m\times n} \times B_{n\times p} = C_{m\times p}$, are mapped to the SideDRAM architecture as depicted in Figure 6. The elements of matrix A are stored as CSD-coded scalars in the CSD register file as multipliers. Meanwhile, horizontal slices of the matrix B are stored in the VWRs, containing as many elements as needed to fill the datapath according to the initial subword width. Next, each element of the CSD RF is multiplied by the corresponding slice in the VWRs, performing vector by scalar multiplication. The resulting vector products are then sequentially added, following a software-defined binary tree scheduling structure. During these additions, subword lengths grow progressively as greater bitwidths are needed in accumulation. The tree structure ensures that additions are done between vectors with matching bitwidths. Ultimately, the final result corresponds with a slice of matrix C, stored in one of the VWRs. If the elements of B have an initial subword with s_B, the resulting elements of C will have a final bitwidth of $s_C = s_B + \lceil \log_2 n \rceil$.

5.2 Simulation Setup

To assess the area, performance, and energy characteristics of SideDRAM, we explore different architectural configurations and compare them with the FIMDRAM design [29], as it represents a proven solution targeting the same domain-specific set of applications. We also compare with TULIP-DRAM [42], a recent CnM mixed-signal solution targeting a similar row-wide interface to DRAM. The parameters of the evaluated configurations are described in Table 2. We explore four configurations of SideDRAM: ***192B***, ***384B***, ***768B***, and ***1536B***, named after the total datapath width per channel (in bytes). All the configurations keep the same ratio between the total widths of the datapath and VWRs. The resulting exploration illustrates the PPA (power, performance, and area) effects of integrating a datapath per sixteen, eight, four, or two banks. Among these, SideDRAM-*384B* provides a direct comparison to FIMDRAM, as it has a similar area occupation. Across the four configurations, the SoftSIMD datapaths support integer subword bitwidths of 3, 4, 6, 8, 12, 16, and 24, with the DPU enabling the increase or decrease of precision between adjacent bitwidths. We assume that all the architectural configurations are integrated into a HBM2 memory [21] as defined in Section 4, and we focus on the behavior of a single channel comprising 16 banks.

Table 2. Parameters of the Explored Baselines and SideDRAM Configurations, Reported Per Channel

Architecture	Configuration	Instruction memory	Data memory	Datapaths	Total number of SIMD lanes
SideDRAM	**192B**	2 × 384 B	2 × 192 B CSDRFs + 2 × 1 kB VWRs	1 × 192 B	64 – 512
	384B	2 × 384 B	2 × 192 B CSDRFs + 4 × 1 kB VWRs	2 × 192 B	128 – 1,024
	768B	2 × 384 B	2 × 192 B CSDRFs + 8 × 1 kB VWRs	4 × 192 B	256 – 2,048
	1536B	2 × 384 B	2 × 192 B CSDRFs + 16 × 1 kB VWRs	8 × 192 B	512 – 4,096
FIMDRAM [29]	FP16	8 × 128 B	8 × 32 B SRFs + 16 × 256 B GRFs	8 × 32 B	128
	INT8	8 × 128 B	8 × 16 B SRFs + 16 × 256 B GRFs	8 × 32 B	256
	INT16	8 × 128 B	8 × 32 B SRFs + 16 × 256 B GRFs	8 × 32 B	128
	INT32	8 × 128 B	8 × 64 B SRFs + 16 × 256 B GRFs	8 × 32 B	64
TULIP-DRAM [42]	FP16	N.A.	N.A.	16,384 × 1 B	16,384
	INT8	N.A.	N.A.	16,384 × 1 B	16,384

The number of lanes refers to how many elements are computed simultaneously. In SideDRAM, this depends on how many subwords of a given bitwidth can be accommodated; in FIMDRAM, it amounts to the number of multipliers and adders implemented for the supported datatype; in TULIP-DRAM, it coincides with the number of mixed-signal ALUs interfaced to the DRAM banks.

In order to evaluate the described SideDRAM configurations, we have developed a CnM exploration framework, available open-source in GitHub[1] composed of a SideDRAM architectural template and a programming interface, similarly to [34]. The architectural template comprises a configurable SystemC model of the SideDRAM PU, including the SoftSIMD datapath, the VWRs and the distributed control plane, and the Ramulator DRAM model [26] extended to support concurrent access to all banks in a channel [34]. In turn, the programming interface allows to translate a program written in assembly code into the sequence of DRAM commands needed to guide CnM execution. For this purpose, we first obtain the assembly program (a sequence of instructions defined by the ISA in Table 1) from the generator in Section 5.1. Then, a custom assembler translates this sequence into the set of DRAM commands needed to program and execute the workload on the architecture, which serves as input to the architectural template. This framework allows to configure (1) the size of the SoftSIMD datapaths, VWRs, CSD, and instruction buffer, (2) the amount of cores per channel, (3) the supported SoftSIMD subword bitwidths and conversion modes, and (4) the format and translation of the control plane instructions and μ-instructions. Thus, fast exploration of the SideDRAM architectural configurations is enabled, providing area, energy, and performance results.

In addition, we employ the CnM framework from [34] to compare to the original FIMDRAM design supporting FP16 computations as a baseline, as well as several integer implementations to showcase a more equitable comparison with the SoftSIMD datapath (Table 2). Particularly, FIMDRAM-*INT8* serves as a proxy to study the performance of the LPDDR-PIM design [25] when executing integer workloads. It should be noted that the integer implementations of FIMDRAM maintain the same precision along the GEMM, thus affecting the quality of service of workloads. Only FIMDRAM-*INT32* can achieve better precision than SideDRAM progressively increasing bitwidths, at the cost of highly decreasing parallelism.

Detailed area and energy of the SideDRAM and FIMDRAM configurations were obtained from DesignCompiler synthesis and PrimeTime power estimation implementing TSMC 28 nm HPC logic technology. All configurations target a frequency of 300 MHz, coinciding with the internal clock frequency of HBM2 [21, 29]. While the SoftSIMD datapath and the VWR components were directly

[1]https://github.com/gem5-X/SideDRAM

described in RTL for synthesis, the distributed control plane and the FIMDRAM configurations were defined in SystemC, which were translated into RTL via Catapult high-level synthesis.

To enable the comparison with TULIP-DRAM, we perform best-case estimations employing the area, power, and latency numbers reported in [42], optimistically assuming that no idle cycles are experienced. For TULIP-DRAM-*INT8*, precision is maintained along the GEMM, as in the FIMDRAM execution. As SideDRAM and FIMDRAM, TULIP-DRAM is integrated into an HBM2 channel, targeting a frequency of 300 MHz. Results are reported for TSMC 40 nm technology, and scaled down to 20 nm [42]. We do not consider the control plane and the data register file for the comparison, as detailed figures are not reported in [42].

5.3 Benchmarks

Throughout our experiments, we analyze the tradeoffs between the PPA benefits of leveraging heterogeneous quantization with the SideDRAM architecture and the corresponding accuracy loss. For this purpose, we employ a set of workloads comprising CNNs and transformers. The CNNs considered are VGG16 [41], MobileNet [18], ResNet20, and ResNet50 [16]. Regarding the transformers, BERT base [11] and DeiT-S [47] are used. The original floating-point benchmarks are executed on FIMDRAM-*FP16* and TULIP-DRAM-*FP16*, while heterogeneously quantized versions are run on SideDRAM and the integer implementations of FIMDRAM and TULIP-DRAM. The quantized workloads are taken from the literature [5, 46, 52], resulting from the application different post-training quantization techniques before execution. Further details details are described in Table 3. To assess how increasing parallelism impacts SideDRAM behavior, we perform batched execution, with 1, 4, and 16 ML inferences run simultaneously.

Performance and energy consumption of the architectures are reported for the fraction of workloads that correspond to GEMM and GEMV operations, which represents the main bulk of their runtime [1, 25, 52]. Each layer of the considered networks is executed as a GEMM, using the mapping scheme for SideDRAM described in section 5.1, and assuming *im2col* transformations for convolutional layers. For the FIMDRAM and TULIP-DRAM configurations, matrix multiplications are mapped in a similar way, decomposed into SIMD scalar by vector MAC operations as described in [34]. In SideDRAM and FIMDRAM, weights are assigned to the CSD or scalar registers, while the activations are divided into vectors for parallel computation using the VWR or vector registers. TULIP-DRAM instead holds both weights and activations in DRAM, replicating the formers for parallel computation [42].

6 Evaluation of SideDRAM on Individual GEMM Kernels

6.1 Area Results

We first assess the area occupation of the SideDRAM configurations to study the overhead at the periphery of the DRAM banks in a channel. Figure 7(a) shows the comparison between SideDRAM, FIMDRAM, and TULIP-DRAM area. SideDRAM-*384B* achieves better area results than FIMDRAM (80%) while supporting a larger total datapath width and similar total data register capacity. These results arise from the better area efficiency of both the multiplierless SoftSIMD datapath and the single-port VWR. In addition, SideDRAM displays a lower control overhead thanks to its shared approach and the resulting better ratio between control elements and datapath elements. In contrast, TULIP-DRAM exhibits significantly greater overhead than FIMDRAM and SideDRAM, with an area occupation 25× larger than FIMDRAM-*FP16*, as it targets extreme parallelism. While individually TULIP-DRAM PUs are very area efficient (0.003× the size of a FIMDRAM-*FP16* lane), this benefit is lost when using many of them to interface to the entirety of the row buffers.

Figure 7(a) also showcases how the area of SideDRAM dedicated to data registers and datapath scales linearly with the VWR capacity and datapath width, respectively, while the control overhead

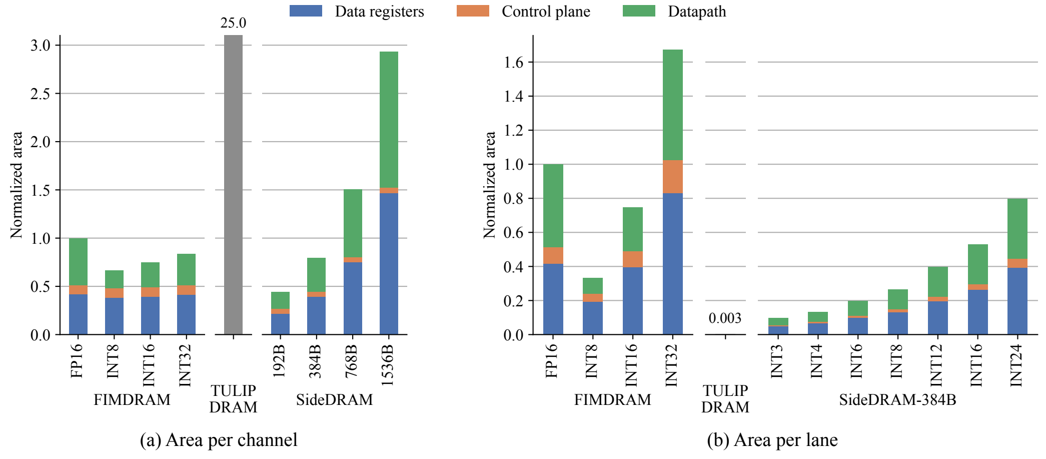

Fig. 7. Post-synthesis area results of the studied SideDRAM configurations integrated in an HBM2 channel, compared to the FIMDRAM and TULIP-DRAM baselines. All results are normalized with respect to FIMDRAM-*FP16*. SideDRAM data registers consist of CSD RFs and VWRs.

is similar across configurations due to the larger elements (instruction buffer, CSD RF, and multiplication sequencer) being shared among PUs. As a result, SideDRAM-*192B* achieves an area 55% lower than FIMDRAM-*FP16*, also smaller than FIMDRAM-*INT8*. On the other hand, SideDRAM-*768B* and *1536B* only increase overhead in 1.51× and 2.93× while multiplying data capacity and computation parallelism by 2 and 4, respectively.

To further analyze the area efficiency of the CnM architectures, Figure 7(b) illustrates area occupation per lane. A lane represents each one of the parallel computations performed simultaneously in the SideDRAM, FIMDRAM, and TULIP-DRAM datapaths. In the first case, a lane coincides with each subword that is computed in parallel. Differently, FIMDRAM lanes coincide with the multipliers and adders implemented in the datapath. TULIP-DRAM lanes instead are equivalent to each PU interfaced to memory. FIMDRAM configurations, each supporting a single data type, present different area efficiency results. Similarly, TULIP-DRAM's area per lane remains constant across different data types, as they can be supported using the same PU. In contrast, SideDRAM area occupation per lane depends on the subword width being employed at runtime. Across bitwidths, SideDRAM efficiency surpasses FIMDRAM-*FP16*, and outperforms FIMRAM-*INT8* up to 8 bits thanks to the avoidance of a dedicated multiplier and the shared control plane. Consequently, SideDRAM is validated as an area-efficient solution under the tightly constrained DRAM bank periphery.

6.2 Performance and Energy Results Across GEMM Sizes and Quantizations

Next, we analyze how workload size and quantization schemes affect the performance and energy consumption of SideDRAM by achieving different utilization rates, that is, the amount of resources that are used for computation in contrast to idle ones. For this purpose, Figure 8 shows speedup and energy results, focusing on the FIMDRAM configurations and SideDRAM-*384B*, when executing GEMMs of different size across the possible initial subword widths. The size of the GEMM is varied by changing the width p of matrix B, which determines whether the workload can be efficiently parallelized across the channel PUs, achieving high resource utilization.

We can differentiate three behaviors in Figure 8: executing small, medium-sized, and large matrices. Small matrices ($p = 64$) incur the underutilization of the FIMDRAM and SideDRAM

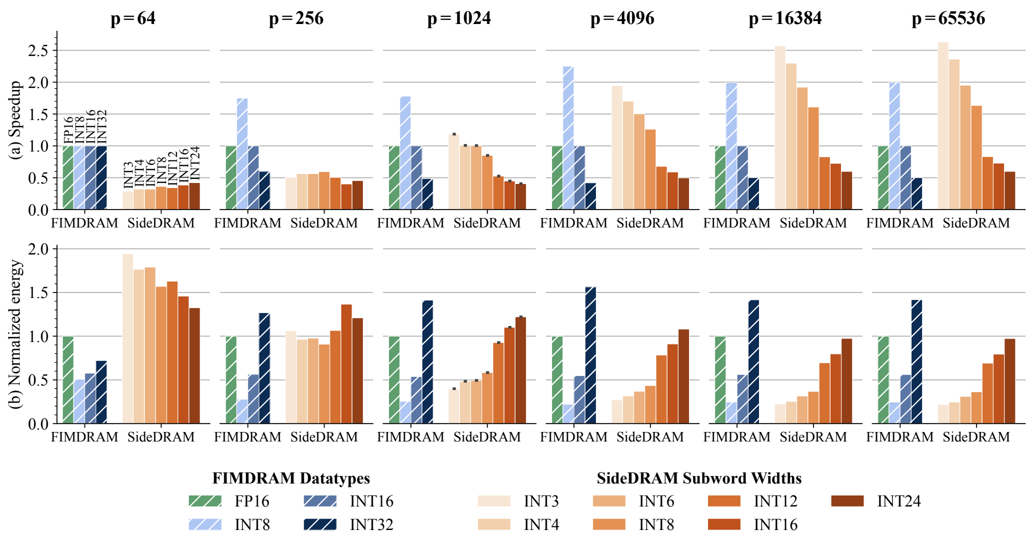

Fig. 8. Speedup (top, the higher the better) and energy (bottom, the lower the better) results when executing GEMM operation of different sizes ($A_{1\times64} \times B_{64\times p}$ for different values of p) and employing different initial subword bitwidths for SideDRAM-*384B*. The bitwidth of the CSD multipliers is 8 bits. Results are normalized with respect to FIMDRAM-*FP16*.

configurations, i.e., the workload is not large enough to utilize all the available resources. The only exception is FIMDRAM-*INT32*, where the 64 SIMD lanes are active. As a result, SideDRAM is slower and less energy efficient due to the lower per-lane performance of the SoftSIMD datapath and the low bandwidth utilization of the VWR interface.

In medium-sized behavior, as observed for $p = 256$ in Figure 8, the computing parallelism and the corresponding utilization depend on the bitwidth used. Indeed, $p = 256$ sets a parallelism level that allows SideDRAM-*384B* to attain high utilization for large bitwidths (e.g., by using all 128 lanes for INT24), but does not achieve the same at smaller bitwidths where more lanes are available (e.g., 1024 lanes for INT3). When executing medium-sized GEMMs SideDRAM performance is improved, but still fails to match FIMDRAM configurations. However, the energy efficiency results start to compare to FIMDRAM for initial subword widths up to INT8. The energy numbers also illustrate how INT24 is still more energy efficient than INT16 for this GEMM size, as the bitwidth cannot be further increased, losing computation accuracy but avoiding the use of the DPUs.

Finally, large matrices ($p = \{1,024,\ 4,096, 16,384, 65,536\}$) achieve full utilization of SideDRAM for any bitwidth. For these workloads, a better performance than FIMDRAM is observed at the smaller subword widths, as the delay from multi-cycle multiplication is offset by the higher parallelism capabilities. We can observe that the workload size sets the speedup rate. At first, larger workloads increase acceleration (between $p = 1,024$ and $p = 16,384$), since the overhead from programming the PUs is offset by the numerous computing iterations required. Then, a zone of diminishing returns is reached ($p > 16,384$), as control overhead become a small fraction of computing time. However, further improvements can be obtained by increasing parallelism, as explored in Section 7. With SideDRAM-*384B*, between 1.63× and 2.63× speedup with respect to FIMDRAM-*FP16* are achieved for initial subwords of INT3 to INT8. These results approach FIMDRAM-*INT8* performance while avoiding precision loss. For wider subwords, performance slowdown is limited to 0.41×, on par with the FIMDRAM-*INT32* configuration which would be needed to maintain precision. SideDRAM high parallelism also improves energy efficiency, achieving up to 78% savings

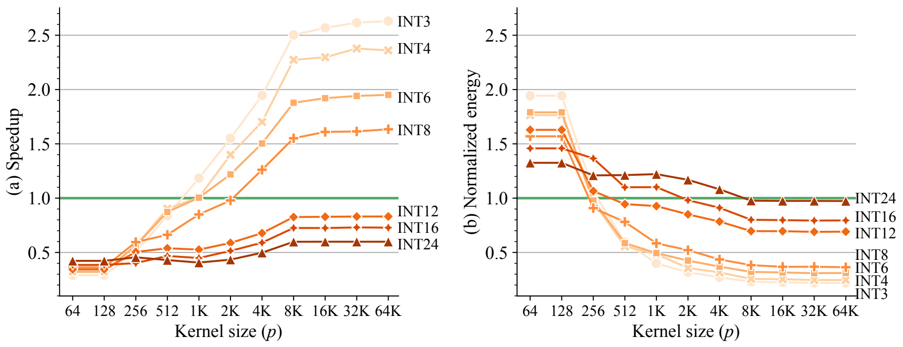

Fig. 9. Evolution of the speedup (left, the higher the better) and energy (right, the lower the better) results when executing GEMM operations ($A_{1\times64} \times B_{64\times p}$) of increasing size for different initial subword bitwidths on SideDRAM-*384B*. The bitwidth of the CSD multipliers is 8 bits. Results are normalized with respect to FIMDRAM-*FP16* for each value of p.

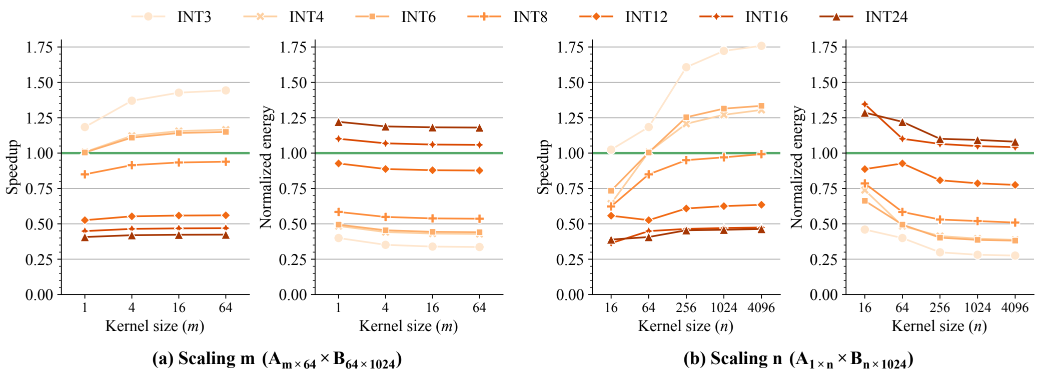

Fig. 10. Evolution of the speedup (left, the higher the better) and energy (right, the lower the better) results when executing GEMM operations ($A_{m\times n} \times B_{n\times p}$) of varying size on SideDRAM-*384B*, increasing the dimensions m (a) and n (b) of the matrices. The bitwidth of the CSD multipliers is 8 bits. Results are normalized with respect to FIMDRAM-*FP16* for each value of m and n.

in subword configurations until INT8, outmatching the efficiency of FIMDRAM-*INT16*. For bigger initial subwords, reasonable energy efficiency is attained despite of the lower performance. Moreover, if the workload is large enough ($p > 4,096$), better energy results are achieved across all the initial subword bitwiths. In the worst case, INT24, a similar efficiency to the one of FIMDRAM-*FP16* is accomplished.

Figure 9 provides a more detailed view of how the speedup and energy gains of SideDRAM-*384B* evolve when increasing the size of the workload for each initial subword width. Between workloads with size $p = 512$ and $p = 2,048$ the configurations up to INT8 progressively overcome FIMDRAM performance, and by $p = 256$ they start surpassing FIMDRAM energy efficiency.

Furthermore, we also analyze how the other dimensions of the GEMM, m and n, impact the performance and energy efficiency of SideDRAM in Figure 10. Results show that the increase of parameter m slightly improves performance and energy efficiency of SideDRAM. When mapping GEMMs to SideDRAM and FIMDRAM (as described in Section 5.1 and [34]) m only sets the number of external loops needed to complete the multiplication. Consequently, the runtime in both architectures grows in a roughly linear fashion with m. Regarding the parameter n, Figure 10(b)

Table 3. Configuration and Representative Results of the Benchmarked Machine Learning Workloads

Network	Dataset	Weight Bits	Activation Bits	Baseline Accuracy	Quantized Accuracy	SideDRAM Speedup	SideDRAM Energy	SideDRAM EDAP
VGG16	CIFAR-100	7, 8	3, 4, 6, 8, 12	69.2%	66.6%	0.89×	0.63×	**0.57**×
MobileNet	CIFAR-100	7, 8	6, 8, 12, 16	64.6%	61.3%	1.37×	0.40×	**0.24**×
ResNet20	CIFAR-100	4, 5, 6, 7, 8	3, 4, 8, 12, 16	70.3%	61.9%	1.57×	0.34×	**0.17**×
ResNet50	CIFAR-100	7, 8	3, 4, 6, 8, 12, 16	75.3%	68.5%	0.85×	0.63×	**0.59**×
BERT-base	MNLI-m	1, 8	8	84.6%	83.3%	1.00×	0.57×	**0.45**×
DeiT-S	ImageNet	4, 5, 6, 7, 8	4, 5, 6, 7, 8	79.8%	76.6%	1.55×	0.35×	**0.18**×

Static quantization schemes and accuracy are taken from state-of-the-art works [5, 46, 52]. Speedup, energy, and EDAP numbers are reported for the execution of inferences with batch size = 16, normalizing the results of SideDRAM-*384B* with respect to FIMDRAM-*FP16*.

shows how its increase enhances SideDRAM behavior, though less prominently than p. Since n determines the length of the dot products composing the GEMM, the resulting larger binary tree structures better offset the control overhead.

In general, the results illustrate how SideDRAM allows to improve energy efficiency for medium-sized and large GEMMs. In addition, steeper improvements are obtained when executing extensive workloads that can leverage the massive parallelism opportunities near the DRAM bank. These energy savings are achieved while maintaining or surpassing the performance of state-of-the-art architectures thanks to the support of heterogeneous quantization and high DRAM bandwidth utilization enabled by the integration of SoftSIMD datapaths and VWRs.

7 PPA Analysis of SideDRAM when Executing AI Benchmarks

The observations above on the characteristics of SideDRAM when executing GEMMs are reflected on the implementation of entire artificial intelligence workloads. In this section, we study the performance, area, and energy tradeoffs offered by the considered SideDRAM configurations by analyzing their behavior when executing AI applications with different batch sizes, i.e., the number of ML inferences performed at once. In the first place, Table 3 shows the parameters of the explored benchmarks, describing the static heterogeneous quantization scheme executed over SideDRAM and the resulting accuracy degradation, as well as the speedup, energy and **energy-delay-area product** (**EDAP**) improvements achieved with SideDRAM-*384B* when compared to FIMDRAM-*FP16*. SideDRAM achieves a similar or better performance than FIMDRAM across benchmarks performing 16-batch inference, while incurring an average accuracy degradation of 4.3%. VGG16 displays the lowest gains, exemplifying the case when the workload does not offer enough opportunities for parallelism. BERT also shows limited speedup as it does not leverage heterogeneous quantization of activations. However, we observe that large enough workloads speed up execution, increasing the performance in up to 1.57× in the case of DeiT-S. The effects of application size are further studied in Section 7.1.

Table 3 also illustrates that high energy savings are achieved by SideDRAM execution, cutting consumption in between 36% and 65%. Overall, the EDAP is reduced across benchmarks when enough parallelism is leveraged, as demonstrated by the geometric mean of the EDAP savings reaching 68%. Section 7.2 showcases the detailed effects of workload size and batching in EDAP.

7.1 Performance and Energy Results

Figure 11 studies the performance and energy consumption when executing the explored inference benchmarks with different batch sizes on the considered architectures. We focus on the MobileNet CNN and the DeiT-S transformer to illustrate the effect of increasing potential workload parallelism

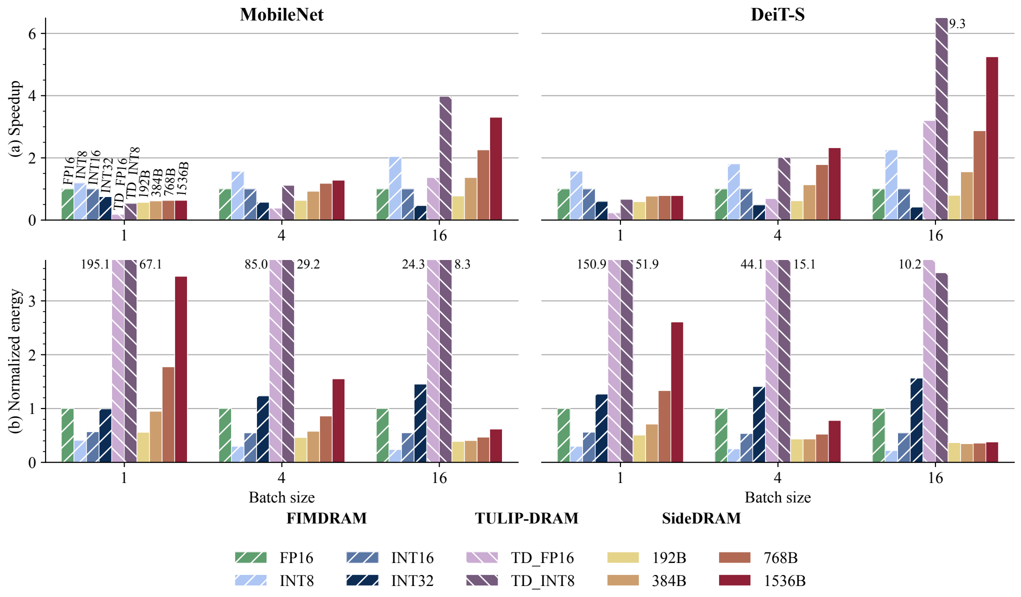

Fig. 11. Speedup (top, the higher the better) and energy (bottom, the lower the better) results when executing MobileNet and DeiT-S inferences with different batch sizes on the explored FIMDRAM and SideDRAM configurations, as well as estimations from TULIP-DRAM [42]. Quantization on SideDRAM is performed according to the activation (subword) and weight (CSD multiplier) bitwidths described in Table 3, configured as in the literature [5, 46, 52]. Results are normalized with respect to FIMDRAM-*FP16* for each workload and batch size.

for execution on SideDRAM through batching. Examining MobileNet inference with one batch showcases that, since the workload does not offer enough parallelism opportunities, execution on SideDRAM-*384B* is 38% slower than on FIMDRAM-*FP16*, while still being 5% more energy efficient. The larger SideDRAM configurations achieve similar runtime degradations, but they increase energy consumption proportionally to their area occupation due to the underutilization of the PUs. TULIP-DRAM also fails to obtain good performance results since a very small fraction of its PUs are effectively utilized. In addition, the high power incurred by the mixed-signal ALUs, together with the low speedups, lead to very high energy overheads.

The performance of the SideDRAM architectures is improved when increasing the inference batch size, as depicted in Figure 11, since it allows to leverage more and more parallelism. Indeed, with a batch size of 16, MobileNet inference is accelerated in up to 3.30× for SideDRAM-*1536B*. Under the same area constraints as FIMDRAM, up to 1.37× improvements are obtained with SideDRAM-*384B*. Moreover, the energy consumption of SideDRAM configurations is a minimum of 38% lower, approaching that of FIMDRAM-*INT8* while maintaining computation precision. In comparison, TULIP-DRAM also benefits from inference batching, showing comparable speedup to SideDRAM with the *FP16* configuration, and superior performance with the less precise *INT8* version. However, while energy efficiency also profits from increasing parallelism, TULIP-DRAM remains an architecture significantly more energy-hungry than FIMDRAM and SideDRAM.

The execution of DeiT-S displays a similar behavior (Figure 11 right), with growing batch sizes resulting in better performance and energy efficiency. In addition, the larger inherent size of GEMM computations in the transformer allows SideDRAM to more steeply increase energy efficiency when using larger batches. This effect is already seen at 1-batch inference, where SideDRAM-*384B* matches FIMDRAM while reducing energy consumption in 29%. As a consequence, significant

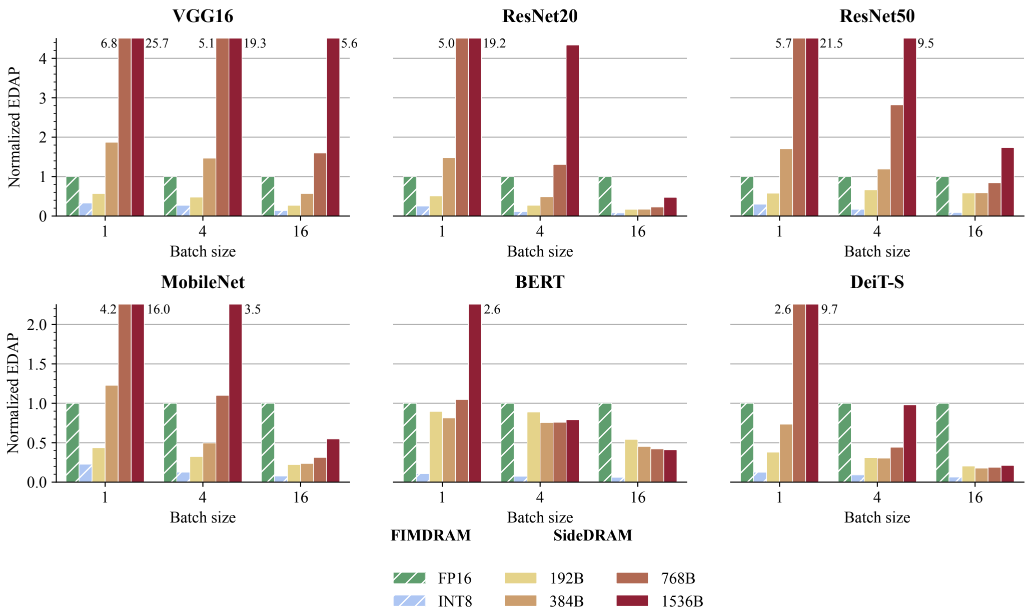

Fig. 12. Energy-delay-area product (EDAP, the lower the better) results when executing inferences of the explored benchmarks with different batch sizes on the FIMDRAM and SideDRAM configurations. Quantization on SideDRAM is performed according to the activation (subword) and weight (CSD multiplier) bitwidths described in Table 3, configured as in the literature [5, 46, 52]. Results are normalized with respect to FIMDRAM-*FP16* for each workload and batch size.

performance and energy efficiency improvements are obtained when executing 16-batch inference, reaching 5.25× speedup and 62% energy savings with SideDRAM-*1536B*. TULIP-DRAM behavior when executing DeiT-S is analogous to MobileNet execution. Leveraging the higher parallelism of the workload, 16-batch inference on TULIP-DRAM-*INT8* surpasses the performance of SideDRAM-*1536B*, but it incurs accuracy loss as FIMDRAM-*INT8*. In addition, the architecture still presents high energy costs.

Across workloads and batch sizes, we observe that SideDRAM-*192B* achieves consistent energy efficiency improvements with respect to FIMDRAM. The configuration accomplishes a geometric mean of 59% energy savings while showing low performance degradation. Nevertheless, all the SideDRAM variations attain high energy efficiency when executing large workloads, at which point the speedup achieved is determined by the parallelism supported by the SoftSIMD datapaths and the VWRs. Thus, employing batch sizes higher than 16 would allow to further decrease runtime and energy consumption until reaching diminishing returns, as shown in Section 6.2. When compared to TULIP-DRAM, the SideDRAM design obtains comparable performance and better energy results for all batch sizes. For small batches, SideDRAM significantly outperforms TULIP-DRAM in performance and energy efficiency, as the latter requires massive parallelism to offset the high latency and power consumption of its computations. Moreover, while TULIP-DRAM can attain higher speedups than SideDRAM for large enough batch sizes, it does so at very high energy overheads.

7.2 EDAP Results

Finally, we assess SideDRAM gains from the joint perspective of performance, energy and area. With this objective, Figure 12 shows the results of the EDAP when executing the explored benchmarks with different inference batch sizes. EDAP results for TULIP-DRAM are not shown, as the possible speedup gains fail to offset the high area and energy overheads. In the best case, i.e., 16-batch BERT, TULIP-DRAM-*FP16* and *INT8* configurations obtain normalized EDAPs of 15.2 and 1.8, respectively, with an efficiency at least 3.3× lower than SideDRAM. Overall, confirming the results from the previous section, we observe that SideDRAM configurations showcase the best EDAP results when executing large workloads that allow to utilize the massive near-DRAM parallelism. In this regard, the use of inference batching is key to achieve high EDAP efficiency with SideDRAM.

Under the same area constraints as in FIMDRAM-*FP16*, SideDRAM also offers lower energy consumption with a low performance degradation. Indeed, SideDRAM-*192B* and *384B* achieve geometric average EDAP reductions in 4-batch inference of 56% and 32%, respectively, thanks to the combination of high area and energy efficiencies.

Although the FIMDRAM-*INT8* implementation showcases the best EDAP across workloads because of its low area and energy overheads, these results do not account for the necessary use of an accumulator to avoid accuracy loss, as discussed in Section 5.2. Instead, SideDRAM configurations can reach similar behavior while allocating the necessary data precision for energy-efficient processing, allowing to choose desired tradeoffs between area occupation and potential speedup. Such tradeoff is reflected in the EDAP results across SideDRAM configurations leveraging wide parallelism, e.g., BERT or DeiT-S 16-batch inference. While for smaller applications SideDRAM-*768B* and *1,536B* display large EDAPs due to underutilization, large workloads allow these configurations to offset their area overhead with performance gains.

8 Conclusion

Current bank-level CnM architectures aiming to address the memory wall fail to exploit the complete bandwidth at the DRAM bank periphery and support a reduced range of datatypes. They thus incur high EDAP as a consequence of the lack in hardware-software co-design. Instead, in this work, we have pointed out software-defined SIMD as a promising CnM strategy enabling the co-optimization of area- and energy-efficient variable precision computations that leverage near-DRAM bandwidth. Integrating such an approach, this article has introduced **SideDRAM**, a novel CnM architecture. It employs SoftSIMD datapaths, which support flexible operand types at runtime with a limited area cost, and can execute MAC operations at a very low power budget. Furthermore, SideDRAM uses asymmetric VWRs to enable an ultra-high bandwidth interface with the DRAM banks, together with a narrower read/write interconnect with the PU. To guide CnM execution, a distributed control plane is employed, shared among several VWRs and datapaths, to achieve low overhead and high scalability. Taking mixed-precision quantized machine learning models as benchmarks, we have demonstrated that SideDRAM outperforms state-of-the-art digital and mixed-signal CnM solutions in terms of area and energy efficiency. Under the same area constraints as the FIMDRAM design, our proposed solution incurs up to 67% reduction in energy costs and 83% savings in EDAP. Moreover, reducing the area constraints of SideDRAM allows to obtain up to 5.25× speedup with respect to FIMDRAM while maintaining 62% energy savings. With respect to the mixed-signal design TULIP-DRAM, SideDRAM attains significant energy efficiency savings, showcasing an average of 15× improvement across benchmarks at less than 12% of the area overhead, while achieving a similar or better performance.

References

[1] Alireza Amirshahi, Joshua Alexander Harrison Klein, Giovanni Ansaloni, and David Atienza. 2023. Tic-sat: Tightly-coupled systolic accelerator for transformers. In *Proceedings of the ASP-DAC*. 657–663.

[2] Algirdas Avizienis. 1961. Signed-digit numbe representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers* EC-10, 3 (1961), 389–400. DOI: https://doi.org/10.1109/TEC.1961.5219227

[3] Mart van Baalen, Andrey Kuzmin, Suparna S. Nair, Yuwei Ren, Eric Mahurin, Chirag Patel, Sundar Subramanian, Sanghyuk Lee, Markus Nagel, Joseph Soriaga, et al. 2023. FP8 versus INT8 for efficient deep learning inference. arXiv:2303.17951. Retrieved from https://arxiv.org/abs/2303.17951

[4] Sathwika Bavikadi, Purab Ranjan Sutradhar, Mark A. Indovina, Amlan Ganguly, and Sai Manoj Pudukotai Dinakarrao. 2024. ReApprox-PIM: Reconfigurable approximate lookup-table (LUT)-based processing-in-memory (PIM) machine learning accelerator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 8 (2024), 2288–2300.

[5] Woohong Byun, Jongseok Woo, and Saibal Mukhopadhyay. 2024. Hardware-friendly hessian-driven row-wise quantization and FPGA acceleration for transformer-based models. In *Proceedings of the ACM/IEEE ISLPED*. 1–6.

[6] Francky Catthoor, Praveen Raghavan, Andy Lambrechts, Murali Jayapala, Angeliki Kritikakou, and Javed Absar. 2010. *Ultra-Low Energy Domain-Specific Instruction-Set Processors*. Springer Science and Business Media.

[7] Seunghwan Cho, Haerang Choi, Eunhyeok Park, Hyunsung Shin, and Sungjoo Yoo. 2020. McDRAM v2: In-dynamic random access memory systolic array accelerator to address the large model problem in deep neural networks on the edge. *IEEE Access* 8 (2020), 135223–135243. DOI: https://doi.org/10.1109/ACCESS.2020.3011265

[8] Jungwoo Choi, Hyuk Jae Lee, and Chae Eun Rhee. 2022. ADC-PIM: Accelerating convolution on the GPU via in-memory approximate data comparison. *IEEE JETCAS* 12, 2 (2022), 458–471. DOI: https://doi.org/10.1109/JETCAS.2022.3167391

[9] Benoît W. Denkinger, Miguel Peón-Quirós, Mario Konijnenburg, David Atienza, and Francky Catthoor. 2022. VWR2A: A very-wide-register reconfigurable-array architecture for low-power embedded devices. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC'22)*. Association for Computing Machinery, New York, NY, USA, 895–900. DOI: https://doi.org/10.1145/3489517.3530980

[10] Fabrice Devaux. 2019. The true processing in memory accelerator. In *Proceedings of the IEEE HCS*.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 4171–4186.

[12] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2022. A survey of quantization methods for efficient neural network inference. In *Proceedings of the Low-Power Computer Vision*. Chapman and Hall/CRC, 291–326.

[13] Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W. Mahoney, and Kurt Keutzer. 2024. AI and memory wall. *IEEE Micro* 44, 3 (2024), 33–39. DOI: https://doi.org/10.1109/MM.2024.3373763

[14] Rishabh Goyal, Joaquin Vanschoren, Victor Van Acht, and Stephan Nijssen. 2021. Fixed-point quantization of convolutional neural networks for quantized inference on embedded platforms. arXiv:2102.02147. Retrieved from https://arxiv.org/abs/2102.02147

[15] Nastaran Hajinazar, Geraldo F. Oliveira, Sven Gregorio, João Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gómez-Luna, and Onur Mutlu. 2021. SIMDRAM: A framework for bit-serial SIMD processing using DRAM. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'21)*. Association for Computing Machinery, New York, NY, USA, 329–345. DOI: https://doi.org/10.1145/3445814.3446749

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE CVPR*.

[17] Mingxuan He, Choungki Song, Ilkon Kim, Chunseok Jeong, Seho Kim, Il Park, Mithuna Thottethodi, and T. N. Vijaykumar. 2020. Newton: A DRAM-maker's accelerator-in-memory (AiM) architecture for machine learning. In *Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 372–385. DOI: https://doi.org/10.1109/MICRO50266.2020.00040

[18] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. Retrieved from https://arxiv.org/abs/1704.04861

[19] Christopher J. Hughes. 2022. *Single-instruction Multiple-data Execution*. Springer Cham. DOI: https://doi.org/10.1007/978-3-031-01746-9

[20] Murali Jayapala, Francisco Barat, Tom Vander Aa, Francky Catthoor, Henk Corporaal, and Geert Deconinck. 2005. Clustered loop buffer organization for low energy VLIW embedded processors. *IEEE Transactions on Computers* 54, 6 (2005), 672–683.

[21] JEDEC. 2021. High bandwidth memory (HBM) DRAM. JESD235D. https://www.jedec.org/standards-documents/docs/jesd235a

[22] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre Luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the ISCA*.

[23] Liu Ke, Xuan Zhang, Jinin So, Jong-Geon Lee, Shin-Haeng Kang, Sukhan Lee, Songyi Han, YeonGon Cho, Jin Hyun Kim, Yongsuk Kwon, et al. 2022. Near-memory processing in action: Accelerating personalized recommendation with AxDIMM. *IEEE Micro* 42, 1 (2022), 116–127. DOI: https://doi.org/10.1109/MM.2021.3097700

[24] Asif Ali Khan, João Paulo C. De Lima, Hamid Farzaneh, and Jeronimo Castrillon. 2024. The Landscape of Compute-near-memory and Compute-in-memory: A Research and Commercial Overview. arXiv:2401.14428. Retrieved from https://arxiv.org/abs/2401.14428

[25] Jin Hyun Kim, Yuhwan Ro, Jinin So, Sukhan Lee, Shin-haeng Kang, YeonGon Cho, Hyeonsu Kim, Byeongho Kim, Kyungsoo Kim, Sangsoo Park, et al. 2023. Samsung pim/pnm for transfmer based ai: Energy efficiency on pim/pnm cluster. In *Proceedings of the 2023 IEEE Hot Chips 35 Symposium (HCS)*. IEEE Computer Society, 1–31.

[26] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2016. Ramulator: A fast and extensible DRAM simulator. *IEEE Computer Architecture Letters* 15, 1 (2016), 45–49. DOI: https://doi.org/10.1109/LCA.2015.2414456

[27] Stefan Kraemer, Rainer Leupers, Gerd Ascheid, and Heinrich Meyr. 2007. SoftSIMD—exploiting subword parallelism using source code transformations. In *Proceedings of the 2007 Design, Automation and Test in Europe Conference and Exhibition*. 1–6. DOI: https://doi.org/10.1109/DATE.2007.364485

[28] Yongkee Kwon, Kornijcuk Vladimir, Nahsung Kim, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Guhyun Kim, Byeongju An, Jeongbin Kim, Jaewook Lee, Ilkon Kim, Jaehan Park, Chanwook Park, Yosub Song, Byeongsu Yang, et al. 2022. System architecture and software stack for GDDR6-AiM. In *Proceedings of the IEEE HCS*.

[29] Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, Jinhyun Kim, Seongil O, Anand Iyer, David Wang, Kyomin Sohn, and Nam Sung Kim. 2021. Hardware architecture and software stack for PIM based on commercial DRAM technology. In *Proceedings of the ISCA*.

[30] Shuangchen Li, Alvin Oliver Glova, Xing Hu, Peng Gu, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. 2018. SCOPE: A stochastic computing engine for DRAM-based in-situ accelerator. In *Proceedings of the MICRO*. 696–709. DOI: https://doi.org/10.1109/MICRO.2018.00062

[31] Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. 2017. DRISA: A DRAM-based reconfigurable in-situ accelerator. In *Proceedings of the 2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 288–301. DOI: https://doi.org/10.1145/3123939.3123977

[32] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware weight quantization for on-device llm compression and acceleration. In *Proceedings of Machine Learning and Systems* 6 (2024), 87–100. Retrieved from https://proceedings.mlsys.org/paper_files/paper/2024/file/42a452cbafa9dd64e9ba4aa95cc1ef21-Paper-Conference.pdf

[33] Haifeng Liu, Long Zheng, Yu Huang, Chaoqiang Liu, Xiangyu Ye, Jingrui Yuan, Xiaofei Liao, Hai Jin, and Jingling Xue. 2023. Accelerating personalized recommendation with cross-level near-memory processing. In *Proceedings of the ISCA*.

[34] Rafael Medina, Giovanni Ansaloni, Marina Zapater, Alexandre Levisse, Saeideh Alinezhad Chamazcoti, Timon Evenblij, Dwaipayan Biswas, Francky Catthoor, and David Atienza. 2024. Bank on compute-near-memory: Design space exploration of processing-near-bank architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 11 (2024), 4117–4129. DOI: https://doi.org/10.1109/TCAD.2024.3442989

[35] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. 2021. A white paper on neural network quantization. arXiv:2106.08295. Retrieved from https://arxiv.org/abs/2106.08295

[36] Geraldo F. Oliveira, Ataberk Olgun, Abdullah Giray Yağlıkçı, F. Nisa Bostancı, Juan Gómez-Luna, Saugata Ghose, and Onur Mutlu. 2024. MIMDRAM: An end-to-end processing-using-DRAM system for high-throughput, energy-efficient and programmer-transparent multiple-instruction multiple-data computing. In *Proceedings of the 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 186–203. DOI: https://doi.org/10.1109/HPCA57654.2024.00024

[37] Abdelkrim Kamel Oudjida. 2014. *Binary Arithmetic for Finite-Word-Length Linear Controllers: MEMS Applications*. Ph.D. Dissertation. L'Université de Franche-Compté.

[38] Sang-Soo Park, KyungSoo Kim, Jinin So, Jin Jung, Jonggeon Lee, Kyoungwan Woo, Nayeon Kim, Younghyun Lee, Hyungyo Kim, Yongsuk Kwon, Jinhyun Kim, Jieun Lee, YeonGon Cho, Yongmin Tai, Jeonghyeon Cho, Hoyoung Song, Jung Ho Ahn, and Nam Sung Kim. 2024. An LPDDR-based CXL-PNM platform for TCO-efficient inference of transformer-based large language models. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 970–982. DOI: https://doi.org/10.1109/HPCA57654.2024.00078

[39] Jorn Peters, Marios Fournarakis, Markus Nagel, Mart Van Baalen, and Tijmen Blankevoort. 2023. Qbitopt: Fast and accurate bitwidth reallocation during training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1282–1291.

[40] Praveen Raghavan, Andy Lambrechts, Murali Jayapala, Francky Catthoor, Diederik Verkest, and Henk Corporaal. 2007. Very wide register: An asymmetric register file organization for low power embedded processors. In *Proceedings of the 2007 Design, Automation and Test in Europe Conference and Exhibition*. IEEE, 1–6.

[41] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556. Retrieved from https://arxiv.org/abs/1409.1556

[42] Gian Singh, Ayushi Dube, and Sarma Vrudhula. 2024. A high throughput, energy-efficient architecture for variable precision computing in DRAM. In *Proceedings of the 2024 IFIP/IEEE 32nd International Conference on Very Large Scale Integration (VLSI-SoC)*. 1–6. DOI: https://doi.org/10.1109/VLSI-SoC62099.2024.10767834

[43] Aman Sinha, Huei Chun Yang, Pei Yi Liu, Yen Shi Kuo, Yuhao Fang, Tien Shuo Chang, Ke Han Li, and Bo Cheng Lai. 2022. DSIM: Distributed sequence matching on near-DRAM accelerator for genome assembly. *IEEE JETCAS* 12, 2 (2022), 486–499. DOI: https://doi.org/10.1109/JETCAS.2022.3172774

[44] Chirag Sudarshan, Mohammad Hassani Sadi, Lukas Steiner, Christian Weis, and Norbert Wehn. 2022. A critical assessment of DRAM-PIM architectures-trends, challenges and solutions. In *Proceedings of the International Conference on Embedded Computer Systems*. 362–379.

[45] Chirag Sudarshan, Taha Soliman, Cecilia De La Parra, Christian Weis, Leonardo Ecco, Matthias Jung, Norbert Wehn, and Andre Guntoro. 2021. A novel DRAM-based process-in-memory architecture and its implementation for CNNs. In *Proceedings of the ASP-DAC*. 35–42. DOI: https://doi.org/10.1145/3394885.3431522

[46] Yu-Shan Tai, An-Yeu, and Wu. 2024. *MPTQ-ViT: Mixed-Precision Post-Training Quantization for Vision Transformer*. Retrieved from https://arxiv.org/abs/2401.14895

[47] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers and distillation through attention. In *Proceedings of the International Conference on Machine Learning*. PMLR, 10347–10357.

[48] Ankit Wagle, Gian Singh, Sunil Khatri, and Sarma Vrudhula. 2024. An ASIC accelerator for QNN with variable precision and tunable energy efficiency. *IEEE TCAD* 43, 7 (2024), 2057–2070. DOI: https://doi.org/10.1109/TCAD.2024.3357597

[49] Ruonan Wang, Rodrigo Tobar, Markus Dolensky, Tao An, Andreas Wicenec, Chen Wu, Fred Dulwich, Norbert Podhorszki, Valentine Anantharaj, Eric Suchyta, Baoqiang Lao, and Scott Klasky. 2020. Processing full-scale square kilometre array data on the summit supercomputer. In *Proceedings of the SC*.

[50] William A. Wulf and Sally A. McKee. 1995. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH Computer Architecture News* 23, 1 (1995), 20–24.

[51] Tongxin Xie, Zhenhua Zhu, Bing Li, Yukai He, Cong Li, Guangyu Sun, Huazhong Yang, Yuan Xie, and Yu Wang. 2025. UniNDP: A unified compilation and simulation tool for near DRAM processing architectures. In *Proceedings of the 2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*.

[52] Pengbo Yu, Flavio Ponzina, Alexandre Levisse, Mohit Gupta, Dwaipayan Biswas, Giovanni Ansaloni, David Atienza, and Francky Catthoor. 2024. An energy efficient soft SIMD microarchitecture and its application on quantized CNNs . *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 32, 06 (2024), 1018–1031. DOI: https://doi.org/10.1109/TVLSI.2024.3375793

Received 5 August 2025; revised 5 August 2025; accepted 7 August 2025