FePIM: Contention-Free In-Memory Computing Based on Ferroelectric Field-Effect Transistors

Xiaoming Chen*#, Yuping Wu#, Yinhe Han
#Equal contribution. *Corresponding author (chenxiaoming@ict.ac.cn).
Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China

ABSTRACT

The memory wall bottleneck has caused a large portion of the energy to be consumed by data transfer between processors and memories when dealing with data-intensive workloads. By giving some processing abilities to memories, processing-in-memory (PIM) is a promising technique to alleviate the memory wall bottleneck. In this work, we proposed a novel PIM architecture by employing ferroelectric field-effect transistors (FeFETs). The proposed design, named FePIM, is able to perform in-memory bitwise logic and add operations between two selected rows or between one selected row and an immediate operand. By utilizing unique features of FeFET devices, we further propose novel solutions to eliminate simultaneous-read-and-write (SRAW) contentions such that stalls are eliminated. Experimental results show that FePIM reduces 15% of the memory access latency and 44% of the memory access energy, compared with an enhanced version of a state-of-the-art FeFET-based PIM design which cannot handle SRAW contentions.

CCS CONCEPTS

• **Hardware → Non-volatile memory**; **Emerging architectures**.

KEYWORDS

Ferroelectric field-effect transistor, processing-in-memory, data contention

ACM Reference Format:
Xiaoming Chen, Yuping Wu, Yinhe Han. 2021. FePIM: Contention-Free In-Memory Computing Based on Ferroelectric Field-Effect Transistors. In *26th Asia and South Pacific Design Automation Conference (ASPDAC '21), January 18–21, 2021, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3394885.3431530

1 INTRODUCTION

In the past decades, the performance gap between processors and memories has been continuously widened [13], which is known as the memory wall bottleneck. The recent development of big-data applications has further exerted great pressure to the conventional computer architecture. Due to the high cache miss caused by the irregular memory access patterns and the high volumes of data, a huge amount of data needs to be transferred between processors and memories through the narrow off-chip memory bus. It is reported that a data access from the dynamic random-access memory (DRAM) consumes three orders of magnitude higher energy then a simple arithmetic operation [8]. As a result, the energy consumption of data movement can be over 60% and up to over 90% of the total energy consumption for big-data applications [19].

Processing-in-memory (PIM) is a promising technique to alleviate the memory wall bottleneck. By integrating some lightweight processing units into the main memory, the pressure of processor-memory data movement can be dramatically relieved. There are roughly two categories of modern PIM implementations. The first implementation takes advantage of 3D integration and places the processing units on the bottom layer of the 3D stacking (e.g., [1, 5, 14]). The second implementation employs emerging nonvolatile memory devices and modifies the peripheral circuitry to perform in-memory bitwise logic operations without the requirement of 3D integration, resulting in a more cost-efficient solution. In recent years, people have proposed the second-category PIM architectures based on resistive random-access memories (RRAMs) [4, 9], spin-transfer torque random-access memories (STT-RAMs) [10], phase-change memories (PCMs) [12], and ferroelectric field-effect transistors (FeFETs) [15].

Despite the aforementioned nonvolatile device based PIM designs are claimed to achieve high energy efficiency and/or performance, there are several practical and important issues that have never/rarely been considered, potentially preventing applications benefiting from PIM architectures. Most of the existing PIM designs only provide in-memory bitwise logic operations without considering write back. How to efficiently control write back is an inevitable issue. Consider a normal case where a PIM operation is completed and the result needs to be written back. Meanwhile, we need to read some data out for the subsequent PIM operations. The write back and read operations may contend. In a way, this problem is similar to the data hazard issue in the pipeline architecture of processors.

Though Ref. [11] proposed an FeFET-based memory design that supports direct write back in the same clock cycle of PIM operations, it has two major drawbacks. First, the cells that are not being written are $V_{\mathrm{DD}}/2$ biased, putting them in an unstable state as the $V_{\mathrm{DD}}/2$ bias may perturb the ferroelectric polarization. Second, direct write back requires some signals to be switched at some on-the-fly conditions within a clock cycle, which is difficult to implement and vulnerable to variations. Instead, latching the PIM result and writing it back in the next clock cycle is still the most reliable way. However, when scheduling write back, PIM operations and normal memory operations together, practical applications can frequently cause simultaneous-read-and-write (SRAW) contentions. This problem has never been studied in PIM. If such data contentions are not well handled, the PIM processing units have to be stalled until contentions disappear, resulting in performance loss.

In this work, we employ FeFETs to build a contention-free PIM architecture. Compared with RRAMs, STT-RAMs and PCMs, FeFETs have two unique advantages. First, the OFF-state current of FeFETs is negligibly small (experiments have demonstrated 10^{-12}A or even 10^{-15}A OFF-state currents [17, 21]), which helps reduce the energy consumption. More importantly, FeFETs are three-terminal devices and the read and paths are separated. This offers a higher flexibility when designing FeFET-integrated circuits (e.g., multi-functional PIM memories [22]). In addition to designing a PIM architecture which supports in-memory logic and add operations, the data contention problem is introduced and solved by utilizing the separated-read-and-write-paths feature of FeFETs[1]. Our contributions are summarized as follows.

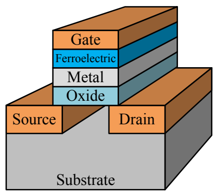
- We propose an FeFET-based PIM architecture named FePIM. Compared with existing PIM designs, FePIM is a complete PIM architecture with controllers, which operate according to an elaborated finite state machine (FSM). The controller not only enables the cooperation of the normal memory operations and PIM operations, but also handles in-memory data contentions.
- We introduce the data contention problem in the FeFET memory array. We design a novel FeFET-based memory array architecture that natively supports **SRAW operations on different rows**.
- We further propose a forwarding mechanism that supports **SRAW operations on the same row**. These are two important features of FePIM to eliminate stalls caused by SRAW contentions.
- As far as we know, this is the first time the data contention problem has been introduced and solved in PIM architectures.

2 PRELIMINARY

An FeFET is made by integrating a ferroelectric material layer in the gate stack of a metal-oxide-semiconductor field-effect transistor (MOSFET), as shown in Fig. 1a. FeFETs are compatible with MOSFETs [18]. The behavior of an FeFET strongly depends on the ferroelectric layer material and thickness. By properly selecting the ferroelectric layer thickness, a hysteresis loop in the I_{DS}-V_G curve can be obtained [6] so as to introduce nonvolatility into the device.

We adopt a SPICE-compatible FeFET simulation model from [3]. Fig. 1b shows a simulated hysteresis curve when the ferroelectric layer thickness is 5.4nm. The existence of the hysteresis curve indicates that the state of an FeFET can be read out by setting a proper V_G (e.g., $V_G = 0$V in the case of Fig. 1b) at a fixed nonzero V_D. In this case, we can obtain two possible values of I_{DS}, i.e., I_{ON} or I_{OFF}, depending on the state of the FeFET (i.e., ON or OFF), which in turn, depends on the ferroelectric layer polarization. When standby, a proper bias voltage (e.g., 0V in the case of Fig. 1b) is needed to the gate terminal of an FeFET to maintain its polarization.

[1]Contention in this paper refers to in-memory contention. Contention between processors and memories, which is the coherence issue [20], is out of the scope of this paper.



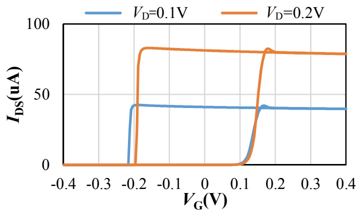


Figure 1: FeFET background. (a) FeFET structure. (b) Simulated hysteresis curve.

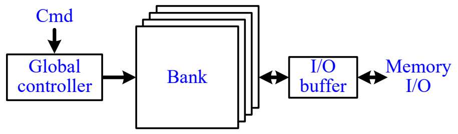


Figure 2: Top-level diagram of FePIM architecture.

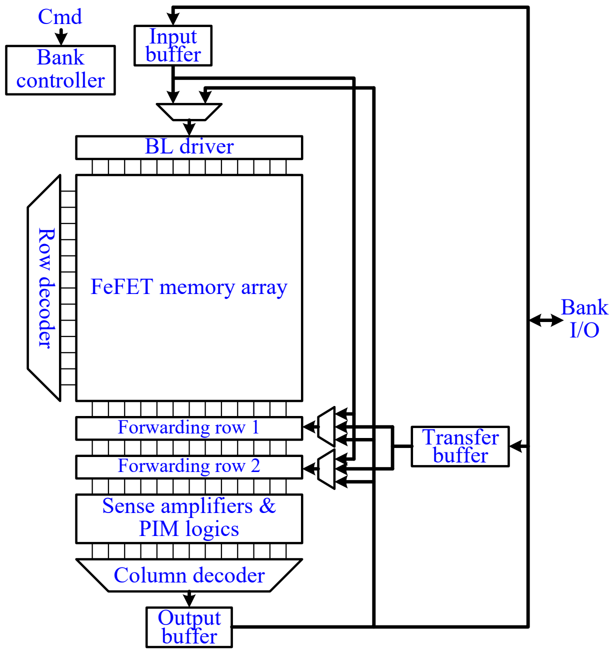


Figure 3: Bank architecture of FePIM.

To change the polarization of the ferroelectric layer and also change the state of an FeFET, we need to supply a positive or negative V_G pulse with a proper magnitude (e.g., 0.4V according to Fig. 1b) to the gate terminal. The applied V_G pulse produces a positive or negative electric field on the ferroelectric layer, and changes its polarization. FeFETs are nonvolatile, meaning that the ferroelectric layer polarization maintains after power off.

3 FEPIM ARCHITECTURE

3.1 Architecture Overview

The top-level architecture of FePIM is shown in Fig. 2. Similar to the conventional memory architecture, FePIM is also constituted by a set of banks. To schedule PIM operations, we add a global

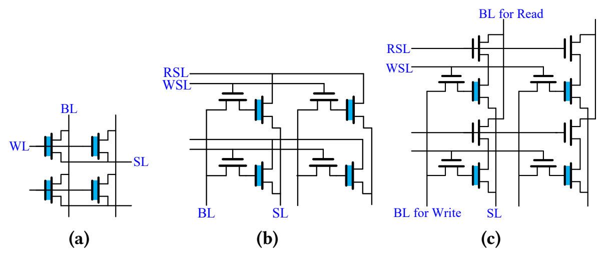


Figure 4: Existing FeFET memory designs (2×2 example). (a) 1T cell [16]. (b) 2T cell [7]. (c) 3T cell [15].

PIM controller in the memory. The global controller receives a PIM command in each clock cycle, decodes the command, and sends the command to the proper bank(s) to execute PIM operations.

Bank is the core component of FePIM. Fig. 3 shows the bank architecture of FePIM. The primary components of a bank are an FeFET memory array, sense amplifiers (SAs) and PIM logics, a bank controller, two forwarding rows (FwRows), and other peripheral circuitry. The bank controller receives commands from the global controller and generates all the control signals as well as the addresses to schedule PIM operations in the bank. The bank controller operates based on an finite state machine (FSM), which is described in Section 3.6. The FeFET memory array has two primary functionalities: memory mode and PIM operation mode. In the memory mode, it works similar to a conventional random-access memory (RAM). In the PIM operation mode, typically two rows are selected (PIM operations between one selected row and an immediate operand are also supported) and the result is produced by the SAs and PIM logics. The SA design and PIM logics are introduced in Section 3.3. We do not explicitly differentiate the two modes because the SAs and other peripheral circuitry are common for both modes. We propose two novel techniques to deal with SRAW contentions so stalls caused by SRAW contentions are completely avoided. The details are described in Sections 3.2 and 3.4.

3.2 FeFET Memory Design

Before introducing our new design, we first briefly review existing FeFET memory designs based on 1T cells [16], 2T cells [7] and 3T cells [15], as shown in Fig. 4. The 1T cell based memory only supports column-wise write and row-wise read, which requires a complex peripheral circuitry. It cannot support SRAW operations due to the fact that the bit line (BL) voltages for read and write operations conflict. For write, the BL corresponding to the writing column is set to V_{DD} and all the other BLs are set to 0. For read, all BLs are set to V_{READ}. This implies that, there is no way to set common BL voltages to perform read and write operations at the same time. Although the 2T cell and 3T cell based memory designs both support row-wise read and write, they cannot support SRAW operations either. The reason is the same. For read, the BLs are set to 0 to maintain the FeFETs' polarizations. For write, the BLs are set to the writing voltages. Consequently, if the read and write operations happen simultaneously, we are not able to set a common set of BL voltages that can support both read and write operations.

Supporting SRAW operations is important for PIM. We consider a simple example which involves two general PIM commands: `a = b op1 c` and `d = e op2 f`, where `op1` and `op2` are two PIM operators. During the first clock cycle, `b` and `c` are read out and `a` is calculated. During the second clock cycle, we need to write back `a` and also read out `e` and `f` for the second command. If the memory cannot support SRAW operations, we have to write back `a` first and stall the second command for one clock cycle. In this case, if we have K such PIM commands, we need $2K$ cycles to finish them, which spends double time of what we expect.

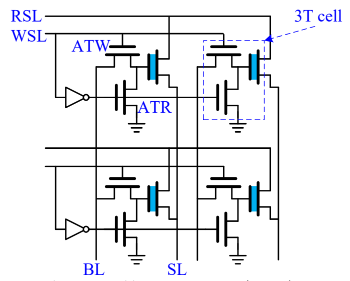


Figure 5: Proposed 3T-cell memory (2×2) supporting SRAW operations on different rows.

Table 1: Voltage settings for memory operations.

	Read		Write		Standby
	Sel. row	Unsel. row	Sel. row	Unsel. row	
RSL	V_{READ}	0	0	X	0
WSL	$-V_{\mathrm{DD}}$	$-V_{\mathrm{DD}}$	V_{DD}	$-V_{\mathrm{DD}}$	0
BL	X		$\pm V_{\mathrm{WRITE}}$		0
SL	0 (for sensing)		0		0

'X' means don't-care.

To overcome this challenge, we propose a novel FeFET memory design based on a new 3T cell design, as shown in Fig. 5. According to the above analysis on the existing FeFET memory designs, the key to support SRAW operations on different rows is to liberate BLs during read, because the BLs may be used for write. However, during read, the gate terminals of the FeFETs must be grounded to maintain the FeFETs' polarizations. Without the functionality of the BLs during read, the gate terminals of the FeFETs will be floating which may lead to wrong read results. Moreover, the leakage currents of the floating gates may lead to polarization loss. To solve this problem, we use two access transistors in each cell. The access transistors for read (ATRs) behave like the conventional access transistors in the existing 2T cell and 3T cell based memory designs. We add the access transistors for write (ATWs) to set the gate voltages of the FeFETs during read. The new FeFET memory design enables SRAW operations on different rows.

For write, only one write select line (WSL) is activated. The BLs are set to $\pm V_{\mathrm{WRITE}}$ according to the writing values. The values will be programmed into the activated row of the FeFETs through the ATWs. For all inactivated rows, since the WSLs are $-V_{\mathrm{DD}}$, the gate terminals of the FeFETs are grounded through the ATRs so that their polarizations are also maintained. For read or PIM operations, we activate one or two read select lines (RSLs). The read result or the PIM operation result between two activated rows is read from the sense lines (SLs) through the SAs. The voltage settings for memory operations are shown in Table 1.

Now we consider SRAW operations on different rows. For the row selected to write, its RSL is grounded. Hence, V_{DS} of the FeFETs in the writing row is always 0. This implies that the write operation will not affect the SL currents. For the one or two rows selected to

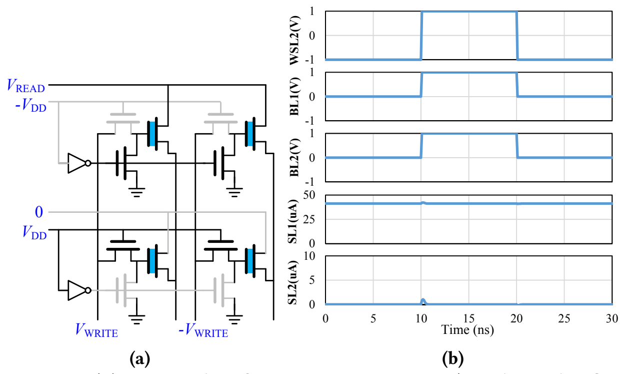


Figure 6: (a) Example of SRAW operations (read on the first row and write on the second row). (b) Simulated waveforms of SRAW operations.

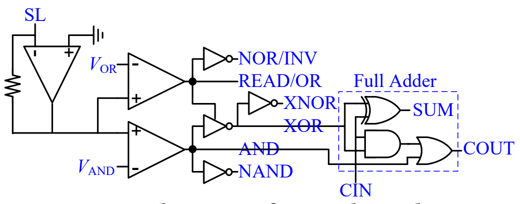


Figure 7: Schematic of SA and PIM logics.

read, since their WSLs are set to $-V_{DD}$, the gate terminals of the FeFETs are grounded through the ATRs to maintain their polarizations. Fig. 6a illustrates an example of SRAW operations. Fig. 6b shows the simulated waveforms of this case. The two FeFETs in the first row are ON and OFF respectively. From 10ns, we begin to program the two FeFETs in the second row both to ON. The simulated waveforms show that the SL currents of the first row are not affected by the programming of the second row.

The success of supporting SRAW operations on different rows comes from two facts: the liberation of the BLs during read and the separated-read-and-write-paths feature of FeFETs. Due to the latter, any two-terminal device cannot apply our approach since the read and write paths are the same. Compared with the existing memory designs without the support of SRAW operations, our design reduces the clock cycles of PIM operations by half at most. However, the current design cannot read and write the same row at the same time. We propose a simple yet effective solution in Section 3.4.

3.3 PIM Operations

FePIM can perform some bitwise logic operations and an add operation between two memory rows or between one memory row and an immediate operand. To realize PIM operations, one or two memory rows are activated. The activated rows will contribute currents to the SLs. Like the existing nonvolatile device based PIM architectures [4, 9, 10, 12, 15], we also modify the SAs to sense the SL currents. Our SA design is shown in Fig. 7. An operational amplifier is used to clamp the SL voltage. The current output is converted to voltage and then fed into two voltage comparators with two reference voltages: V_{OR} for sensing OR/NOR/INV/memory read and V_{AND} for sensing AND/NAND. The XOR/XNOR outputs are

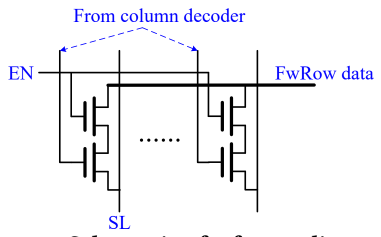


Figure 8: Schematic of a forwarding row.

produced by utilizing the AND and OR outputs at the same time. By utilizing the XOR and AND outputs, a full adder is trivial to construct. The output of the PIM logics is selected from all the logic and adder outputs according to the PIM command.

3.4 Dealing With Data Contention by Forwarding Rows

The FePIM architecture introduced till now solves the SRAW contention problem on different rows, but cannot support SRAW operations on the same row. SRAW operations on the same row can also happen in practical applications. We consider another simple example which involves two general PIM commands: `a = b op1 c` and `d = a op2 e`. In the second clock cycle, we write back the result of the first command `a`, and at the same time, we intend to read out `a` for the second command, bringing SRAW operations on the same row. Without supporting SRAW operations on the same row, we need to stall FePIM by one clock cycle. We introduce a simple yet effective forwarding mechanism as follows.

As shown in Fig. 3, we add two FwRows at the bottom of the memory array of each bank. The schematic of a FwRow is illustrated in Fig. 8. The FwRows work as follows. If an SRAW contention happens like in the above case, in the second cycle, the result of the first command, `a`, which is stored in the output buffer, is forwarded into one FwRow. `a` is written back to the memory in the second cycle. Meanwhile, the second command can be executed by reading the FwRow and the row that stores `e`. An activated FwRow behaves like a memory row. In other words, an activated FwRow contributes to the SL currents. With the FwRows, we realize an equivalent effect of executing SRAW operations on the same row. Combined with the FeFET-based 3T cell design, SRAW contentions are completely avoided in FePIM, which is never achieved by existing nonvolatile memory based PIM architectures.

As can be seen from Fig. 3, the inputs of the FwRows can be selected as the input of a bank. In this case, FePIM can perform PIM operations between a selected row and an immediate operand. This is the secondary functionality of the FwRows, which is not supported by existing nonvolatile device based PIM architectures. It is also possible that two FwRows are both activated. Consider the following two PIM commands: `a = b op1 c` and `d = a op2 0x12345678` (`0x12345678` is a representation of an immediate operand). In this case, in the second cycle, the first FwRow selects `a` as input, and the second FwRow selects the immediate operand, which is the input of the bank, as input.

3.5 Inter-Bank Data Movement

Data movement is needed when the two operands of a command are not in the same bank. A data movement operation takes two

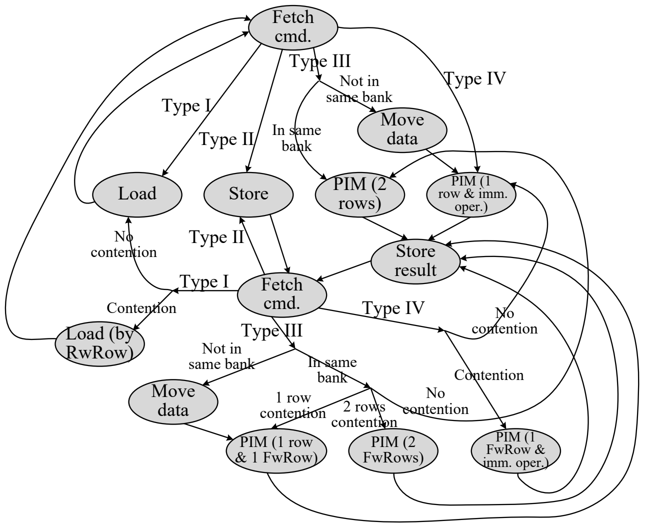


Figure 9: FSM of bank controller.

clock cycles. In the first cycle, the source bank reads the data to be moved, which is written into the transfer buffer of the destination bank in the second cycle. In the following cycles, in the destination bank, the operand is fed into one FwRow and can be used for PIM operations. Data movement brings inevitable pipeline stall. But note that this is not a unique issue of FePIM. The previous PIM architectures [4, 9, 10, 12, 15] all have the same issue. A possible solution to reduce data movement is to optimize the data location in the memory, which is out of the scope of this paper.

3.6 FSM of Bank Controller

The bank controller operates according to an FSM. The bank controller not only schedules the memory load and store operations and PIM operations, but also deals with SRAW contentions and data movement. All PIM commands are classified into four types: I) memory load, II) memory store, III) PIM operation between two rows, and IV) PIM operation between one row and an immediate operand.

The proposed FSM is illustrated in Fig. 9. Once a command is fetched, the subsequent actions are determined based on the types of the current and previous commands. If the previous command is of type I, namely, memory load, then no SRAW contention can happen. In this case, the command is dispatched and executed normally. On the other hand, once a store operation (either a store command or a write back of a PIM operation) is finished, we need to check if there are SRAW contentions for the next command, if the next command involves any read operation. If so, one or two FwRows will be activated to handle the SRAW contentions. For type III commands, we need to first check if the two operands are in the same bank. If not, data movement is needed.

4 EVALUATION RESULTS

FePIM is simulated with HSPICE using the FeFET model proposed in [3]. The 45nm predictive technology model [2] is used as the basic MOSFET model. We use the following voltage settings: V_{DD} = 1V, V_{READ} = 0.1V and V_{WRITE} = 1V. The baseline for comparison is an enhanced version of a state-of-the-art FeFET-based PIM design [15]. The memory array of the baseline is based on the existing 3T cell design (Fig. 4c). We enhance [15] by adding the write back functionality, but SRAW contentions cannot be avoided in the baseline.

Table 2: Comparison on energy consumption (in pJ) of basic operations of a 1MB array with 32-bit word size.

	Read	Write	PIM operation	SRAW
[15]	45.65	45.02	75.72	90.07
Our design	59.63	63.57	79.35	**65.01**

Table 3: Memory access breakdown (%).

	NCWs	CWs	NCRs	CRs	ContRs	RwIOs
MA	0	33.33	0	66.67	33.33	0
HIST	0	33.33	0	66.67	33.33	66.67
QSORT	1.46	22.06	1.45	75.03	0	54.43
RSORT	12.5	25	0	62.5	25	50
XORENC	0	50	0	50	50	50
AES	9.89	23.08	13.46	53.57	26.79	4.95
KMP	0.14	0	1.81	98.05	0	0
KNAPSACK	0	20.04	39.98	39.98	20.04	0
DIJKSTRA	0.13	0.09	0.03	99.75	0.17	36.24
FLOYD	2.04	2.04	0	95.92	2.04	31.97

4.1 Operation-Level Evaluation

We first perform an apple-to-apple comparison with [15] on the energy consumption of basic operations. We also evaluate a 1MB memory array with 32-bit word size (same as [15]). Table 2 lists the energy consumption of read, write, PIM and SRAW operations. For a single operation (a read, write or PIM operation), our memory array consumes a little higher energy than the design of [15]. The higher energy consumption mainly comes from the static power of the inverters in each row of our memory array (see Fig. 5) and the operational amplifiers in the SAs. However, if we consider an SRAW operation, our memory array can reduce the energy by 28%, since the memory design of [15] does not support SRAW operations so it takes two clock cycles to finish an SRAW operation.

4.2 System-Level Evaluation

Here we perform evaluations on the entire FePIM architecture, where the analog components are evaluated by HSPICE and the controllers are implemented by Verilog and evaluated by Design Compiler. The bank size is 1024×1024 and we assume 8 banks in FePIM. The clock frequency is 500MHz. The baseline has the same memory size for a fair comparison. We use 10 benchmarks which are implemented by C to conduct a system-level evaluation. The 10 benchmarks are from different areas, including matrix add (MA), histogram (HIST), quick sort (QSORT), radix sort (R-SORT), XOR encryption (XORENC), advanced encryption standard (AES), Knuth-Morris-Pratt string matching algorithm (KMP), 0-1 knapsack problem (KNAPSACK), Dijkstra shortest path algorithm (DIJKSTRA), and Floyd shortest path algorithm (FLOYD). The computation-to-communication ratio of the 10 benchmarks is low so they are suitable for PIM architectures.

Table 3 lists the memory access breakdown of the 10 benchmarks. Like the methodology described in [15], memory accesses are classified into convertible writes (CWs), non-convertible writes (NCWs), convertible reads (CRs) and non-convertible reads (NCRs). For each benchmark, the sum of these four columns is 100%. Since

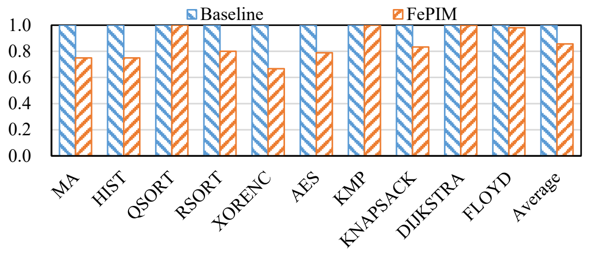


Figure 10: Normalized memory access latency comparison.

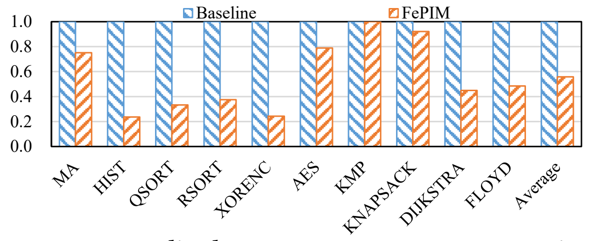


Figure 11: Normalized memory access energy comparison.

FePIM can handle SRAW contentions and PIM commands with immediate operands, we also list the percentages of contending reads (ContRs) and reads with immediate operands (RwIOs) in the last two columns. CRs refer to those memory reads that can be converted to a PIM command, i.e., two successive reads with an operation or one read followed by an operation with an immediate operand. ContRs refer to those memory reads that contend with previous write operations. CWs refer to those memory writes that are associated with PIM operations.

Fig. 10 shows the system-level normalized memory access latency comparison. FePIM reduces 15% of the memory access latency on average compared with the baseline. The memory access latency reduction is due to the elimination of stalls caused by SRAW contentions. For benchmarks QSORT, KMP, DIJKSTRA and FLOYD, the memory access latency reduction is negligible due to their small or even zero ContR rates, as listed in Table 3.

Fig. 11 shows the system-level memory access energy comparison. FePIM reduces the memory acess energy consumption by 44% on average. The memory access energy reduction comes from 1) the elimination of the static power consumed by stalling cycles which are in turn caused by SRAW contentions and 2) the lower energy consumption of PIM operations with immediate operands. The latter dominates the memory access energy reduction. For benchmarks KMP, the memory access energy reduction is zero due to the fact that its CoutRs and RwIO rates are both 0, as listed in Table 3.

5 CONCLUSION

PIM is a promising technique to alleviate the memory wall bottleneck. Existing nonvolatile device based PIM designs cannot well handle the SRAW contention issue. In this work, we utilize FeFETs to build a contention-free PIM architecture named FePIM. We utilize the unique feature of FeFETs of separated read and write paths to build a novel memory array that natively supports SRAW operations on different rows. Furthermore, a forwarding mechanism is proposed to realize SRAW operations on the same row, so that all possible stalls caused by SRAW contentions are eliminated. Compared with an enhanced version of a state-of-the-art FeFET-based PIM design that cannot handle SRAW contentions, FePIM reduces the memory latency by 15% and the memory energy consumption by 44% on average.

ACKNOWLEDGEMENTS

This work was supported in part by National Key R&D Program of China (No. 2020AAA0105200), in part by Key Research Program of Frontier Sciences, Chinese Academy of Sciences (No. ZDBS-LY-JSC012), in part by Strategic Priority Research Program of Chinese Academy of Sciences (No. XDB44000000), in part by National Natural Science Foundation of China (Nos. 61804155 and 61834006), in part by the Youth Innovation Promotion Association CAS, in part by the Young Elite Scientists Sponsorship Program by CAST (No. 2018QNRC001), and in part by Beijing Academy of Artificial Intelligence (BAAI).

REFERENCES

[1] J. Ahn et al. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *ISCA*. 105–117.
[2] ASU. 2011. Predictive Technology Model. http://ptm.asu.edu/
[3] A. Aziz et al. 2016. Physics-Based Circuit-Compatible SPICE Model for Ferroelectric Transistors. *IEEE EDL* 37, 6 (June 2016), 805–808.
[4] P. Gaillardon et al. 2016. The Programmable Logic-in-Memory (PLiM) computer. In *DATE*. 427–432.
[5] M. Gao et al. 2017. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *ASPLOS (ASPLOS '17)*. 751–764.
[6] S. George et al. 2015. Ncfet based logic for energy harvesting systems. In *SRC TECHCON*.
[7] S. George et al. 2016. Nonvolatile memory design based on ferroelectric FETs. In *DAC*. 1–6.
[8] S. Han et al. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *ISCA*. 243–254.
[9] M. Imani et al. 2017. MPIM: Multi-purpose in-memory processing using configurable resistive memory. In *ASP-DAC*. 757–763.
[10] S. Jain et al. 2018. Computing in Memory With Spin-Transfer Torque Magnetic RAM. *IEEE TVLSI* 26, 3 (March 2018), 470–483.
[11] Mingyen Lee et al. 2020. FeFET-Based Low-Power Bitwise Logic-in-Memory with Direct Write-Back and Data-Adaptive Dynamic Sensing Interface. In *ISLPED*.
[12] S. Li et al. 2016. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *DAC*. 1–6.
[13] D. Patterson et al. 1997. A case for intelligent RAM. *IEEE Micro* 17, 2 (March 1997), 34–44.
[14] S. H. Pugsley et al. 2014. NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads. In *ISPASS*. 190–200.
[15] D. Reis et al. 2018. Computing in Memory with FeFETs. In *ISLPED (ISLPED '18)*. 24:1–24:6.
[16] A. Sharma and K. Roy. 2018. 1T Non-Volatile Memory Design Using Sub-10nm Ferroelectric FETs. *IEEE EDL* 39, 3 (March 2018), 359–362.
[17] P. Sharma et al. 2017. Impact of total and partial dipole switching on the switching slope of gate-last negative capacitance FETs with ferroelectric hafnium zirconium oxide gate stack. In *VLSIT*. T154–T155.
[18] M. Trentzsch et al. 2016. A 28nm HKMG super low power embedded NVM technology based on ferroelectric FETs. In *IEDM*. 11.5.1–11.5.4.
[19] H.-W. Tseng et al. 2015. *Gullfoss: Accelerating and Simplifying Data Movement among Heterogeneous Computing and Storage Resources*. Technical Report. University of California, San Diego. http://csetechrep.ucsd.edu/Dienst/UI/2.0/Describe/ncstrl.ucsd_cse/CS2015-1015
[20] S. Xu et al. 2019. CuckooPIM: An Efficient and Less-blocking Coherence Mechanism for Processing-in-memory Systems. In *ASP-DAC*. 140–145.
[21] W. Zhang et al. 2013. Electrical properties of CaxSr1-xBi2Ta2O9 ferroelectric-gate field-effect transistors. *Semiconductor Science and Technology* 28, 8 (2013), 085003.
[22] X. Zhang et al. 2019. FeMAT: Exploring In-Memory Processing in Multifunctional FeFET-Based Memory Array. In *ICCD*. 541–549.