HashPIM: High-Throughput SHA-3 via Memristive Digital Processing-in-Memory

Batel Oved, Orian Leitersdorf, Ronny Ronen, and Shahar Kvatinsky
Andrew and Erna Viterbi Faculty of Electrical and Computer Engineering, Technion – Israel Institute of Technology, Israel
batelov@campus.technion.ac.il, orianl@campus.technion.ac.il, ronny.ronen@technion.ac.il, shahar@ee.technion.ac.il

Abstract—Recent research has sought to accelerate cryptographic hash functions as they are at the core of modern cryptography. Traditional designs, however, suffer from the von Neumann bottleneck that originates from the separation of processing and memory units. An emerging solution to overcome this bottleneck is processing-in-memory (PIM): performing logic within the same devices responsible for memory to eliminate data-transfer and simultaneously provide massive computational parallelism. In this paper, we seek to vastly accelerate the state-of-the-art SHA-3 cryptographic function using the memristive memory processing unit (mMPU), a general-purpose memristive PIM architecture. To that end, we propose a novel in-memory algorithm for variable rotation, and utilize an efficient mapping of the SHA-3 state vector for memristive crossbar arrays to efficiently exploit PIM parallelism. We demonstrate a massive energy efficiency of 1, 422 Gbps/W, improving a state-of-the-art memristive SHA-3 accelerator (SHINE-2) by 4.6×.

Index Terms—Cryptography, SHA-3, processing-in-memory (PIM), stateful logic, memristor.

I. INTRODUCTION

As we enter the era of data-intensive computing across many Internet of Things (IoT) devices, cryptography is emerging as a crucial field for secure communication. At the core of this field are cryptographic hash functions [1] which are fundamental for tasks such as digital signature generation and verification, key derivation, and pseudo-random bit generation [1], [2]. These functions generalize traditional hashing with additional properties aimed at improving security, such as being very infeasible to invert. An emerging state-of-the-art hash function is Secure Hash Algorithm-3 (SHA-3), which exploits techniques such as sponge construction to enhance security [2].

While hashing is traditionally implemented via software, hardware accelerators are emerging to provide unparalleled performance using ASICs [3]–[6], FPGAs [3], [7], and memristive memories (e.g., ReRAM) [8]–[10]. Hardware accelerators benefit from the inherent flexibility for bit-wise accesses that does not exist in CPUs and GPUs. Yet, processing units, including hardware accelerators, are subject to the memory wall; therefore, when hashing large objects stored in memory (or disk), data transfer becomes the bottleneck [8].

An emerging concept to overcome the memory wall is that of Processing-in-Memory (PIM). The fundamental idea of PIM is to shift the computation into the memory, thereby avoiding the data-transfer between the CPU and memory. Recent techniques for PIM involve using the same physical devices for both binary storage and basic digital logic gates, via technologies such as digital memristive memories [11], [12], DRAM [13], SRAM [14], and FeFET [15]. Previous works have sought to utilize this emerging field to accelerate a wide range of applications, including the SHA-3 cryptographic hash function [8]–[10]. Unfortunately, previous SHA-3 designs require specific complex near-array periphery that leads to costly data conversion which diminishes the benefit of PIM. Conversely, we seek to design an in-array algorithm that utilizes the emerging general-purpose memristive Memory Processing Unit (mMPU) [16] without any custom circuitry.

We focus on a digital memristive PIM architecture [11], [12], [16] as this technology has vast potential for large-scale efficient PIM; regardless, the proposed algorithms can be generalized to additional PIM techniques and technologies. Memristors [17] are similar to resistors as they are two-terminal devices, yet they possess a highly unique property: an applied voltage can alter their internal resistance. Therefore, memristors can inherently support storage by representing binary information via their resistance. Interestingly, recent works [18]–[20] have shown that memristors also support basic logic functionality through stateful logic [16], [21]. Thus, memristors inherently enable both memory and digital logic.

In this paper, we propose an efficient in-memory algorithm for SHA-3, using the mMPU [16], that efficiently exploits the vast potential of PIM. We compare our design to other accelerators and demonstrate superior energy efficiency.

II. BACKGROUND

A. Secure Hash Algorithm-3 (SHA-3)

Hash algorithms reduce a large variable-sized input to a small fixed-sized output (hash value) while maintaining a near-uniform output distribution. Cryptography, the field in computer science establishing secure communication, extends the notion of hash functions to cryptographic hash functions [1] that also possess certain properties which enhance security. For example, these functions must be infeasible to invert and slight modifications to the input drastically change the hash value. The Keccak family of cryptographic hash functions was proposed by Bertoni et al. [22], and was later adopted as part of the state-of-the-art SHA-3 standard [2]. This standard proposes four variations for different output lengths; without loss of generality, we discuss SHA3-256 in this paper.

We describe the overall operation of SHA-3, as shown in Fig. 1(a). The input is a message of size $m$ bits which is padded to be of a length that is a multiple of $r$ and then split into $t$ blocks of size $r$ each (e.g., $r = 1088$). The algorithm
proceeds with the absorbing phase that initializes an internal State array.

### Algorithm 1 Keccak-f

**Input:** State array $A[x][y]$ (for all $x, y \in [0, 4]$)

**Output:** State array $A[x][y]$ (for all $x, y \in [0, 4]$)

1. for $i_r = 0, 1, \ldots, 23$ do
   2. **Theta ($\theta$) step:**
      3. $C[x] \leftarrow A[x][0] \oplus \cdots \oplus A[x][4]$ \hspace{1cm} $\forall x \in [0, 4]$
   4. $D[x] \leftarrow C[x-1] \oplus (C[x+1] \ll a 1)$ \hspace{1cm} $\forall x \in [0, 4]$
   5. $A[x][y] \leftarrow A[x][y] \oplus D[x]$ \hspace{1cm} $\forall x, y \in [0, 4]$

   **Rho ($\rho$) step:**
    5. $A[x][y] \leftarrow A[x][y] \ll \alpha \gamma x [y]$ \hspace{1cm} $\forall x, y \in [0, 4]$

   **Pi ($\pi$) step:**
    6. $A[y][2x + 3b] \leftarrow A[x][y]$ \hspace{1cm} $\forall x, y \in [0, 4]$

   **Chi ($\chi$) step:**
    7. $A[x][y] \leftarrow A[x][y] \oplus (A[x+1][y] \land A[x+2][y])$ \hspace{1cm} $\forall x, y$

   **Iota ($\iota$) step:**
    8. $A[0][0] \leftarrow A[0][0] \oplus RC[i_r]^c$

9. end for

*The operator $\ll$ denotes a rotation (cyclic shift),

$b\gamma x [y]$ denotes the constants that define the rotation amount [2],

$RC[i_r]$ denotes the constants for each round [2].

$C[x], D[x]$ are intermediates, and indexing is modulo 5.

---

**B. Memristive Digital Processing-in-Memory (PIM)**

Memristors [17] are rapidly emerging as novel physical devices that inherently support both storage and logic functionalities. Memristors are similar to resistors in that they are two-terminal resistive devices, yet they also possess a unique property: a sufficiently-high current can modify their internal resistance. Therefore, memristors can be utilized for binary information storage (e.g., designating high resistance as logical 0 and low resistance as logical 1) as data is written with a relatively high current and read using a relatively low voltage (measuring the current and deriving the resistance). Memristors are typically connected in crossbar array structures, as shown in Fig. 2(a). Furthermore, memristors inherently support stateful logic [16] in the resistive domain which sets the resistance of an output memristor conditional on the states of the input memristors (e.g., performing NOR) [18]–[20].

The dual functionality of memristors can be exploited towards the memristive Memory Processing Unit (mMPU): a general-purpose memory with massive computational parallelism for bitwise operations, originating from three forms:

- **Row/column parallelism:** By applying voltages on bitlines/wordlines of crossbars, we find that several gates can be performed in parallel within the array itself (when their columns/rows are aligned across different rows/columns) [11], [12], as shown in Fig. 2(a).

- **Partition parallelism:** An overall crossbar may be split into multiple smaller partitions, using transistor switches that divide the bitlines/wordlines, to enable further parallelism [20], [23], [24], as shown in Fig. 2(b). This enables parallel gates within the same row/column, in addition to the parallelism across multiple rows/columns.

- **Crossbar parallelism:** The overall mMPU consists of many crossbar arrays that may operate in parallel [25].
III. HASHPIM ARCHITECTURE

We propose to exploit the vast potential of the mMPU towards an efficient, scalable, and high-throughput SHA-3 algorithm. The benefit of the mMPU over alternative techniques is both due to the ability to efficiently compute precisely where the state vector is stored, and the high efficiency of processing within an array rather than near-array via peripheral circuits.

We detail the proposed architecture within a single crossbar array as the same operations may be performed in parallel across multiple crossbar arrays. Consider a crossbar array as shown in Fig. 3(a), with size of 1024×1024 bits, divided into 378 partitions (27 horizontally, 14 vertically). Each partition (group of 72×37 memristors) is designated a SHA-3 unit and is assigned to compute the SHA-3 hash for a specific message. Thus, the 378 units enable the parallel computation of SHA-3 on 378 different messages within the same crossbar array.

Each SHA-3 unit stores the state-vector corresponding to that message (throughout the computation) and is responsible for applying the Keccak-f function on the state-vector. We choose to map the 5×64 state-vector onto 25×64 memristors (similar to [9]) as an analysis of the routines in the Keccak-f function revealed that this mapping supports optimal parallelism for the θ, π, χ and ɛ steps. Regarding the ρ step, previous works have been unable to implement this step within the array due to the different rotation amount provided in this step, previous works have been unable to implement this step (see Fig. 1(c)), and have thus instead required a copy of the state-vector, we analyze the cyclic (cyclic shifting) in Algorithm 2. To remain in-place (i.e., not require a copy of the state-vector), we analyze the cyclic dependencies in the rotation and execute them serially with a single slice of redundancy for the storage of the state array.

In this section, we detail the proposed design within each SHA-3 unit for the various steps that comprise Algorithm 1.

A. Theta (θ) Step

We start by reducing the state-vector (see Fig. 1(c)) across the y dimension using exclusive-or (XOR) operations; that is, we compute the XOR of every five columns from the 25×64 state-vector to result in a 5×64 vector representing C[0], . . . , C[4] (see Algorithm 1) that is stored in the intermediate area (see Fig. 4(a)). We compute D[0], . . . , D[4] by copying C[0], . . . , C[4] to 5 additional intermediate columns (in-row gates), shifting all of the columns once (in-column gates), and then computing the XOR with C[0], . . . , C[4]. Lastly, the original state-vector is updated by computing the XOR of all columns with the relevant lanes from D[0], . . . , D[4] (e.g., D[0] is XORed with the first five columns in the state-vector).

B. Rho (ρ) Step

At the core of this step is a variable rotation: for each lane in the state vector, we need to rotate that lane by a given number of bits (determined according to r[x][y] see [2]). The difficulty in this variable rotation is that, unlike the constant shift in θ, every lane may contain a different rotation amount [13]. This is seemingly an inherent contradiction to the operation of the mMPU as the operation involves data-dependent control flow.

A similar task of in-array variable shifting was recently considered for in-memory floating-point operations [24]. Essentially, the variable shift is represented as a sequence of multiplexer operations that are chosen according to a logarithmic-shifter [26] design, thereby converting the control-flow to dataflow. We extend this algorithm to the task of variable rotation (cyclic shifting) in Algorithm 2. To remain in-place (i.e., not require a copy of the state-vector), we analyze the cyclic dependencies in the rotation and execute them serially with a single slice of redundancy for the storage of the state array.

We store the constant shift amounts (r[x][y]) at the bottom of the crossbar array (ROT in Fig. 3(a)), shared across all of the units that are aligned vertically (to improve memory utilization for a negligible latency increase). Therefore, as shown in Fig. 4(b), we find that each iteration of Algorithm 2 begins by retrieving the relevant bit for the shift amount from the shared ROT for that vertically-aligned set of units.
The expressions for throughput and power are:

\[ T_{\text{put}}(\text{Unit}) = \frac{r}{\text{Latency}_{\text{Round}}} \times f, \]
\[ T_{\text{put}}(\text{System}) = T_{\text{put}}(\text{Unit}) \times U_{XB} \times N_{XB}, \]
\[ P_{\text{Power}}(\text{System}) = \frac{T_{\text{put}}(\text{System}) \times \text{Energy}_{\text{unit}}}{r}. \]

Table I summarizes our results and compares to other SHA-3 accelerators (both CMOS-based and memristor-based). The code repository [27] includes additional results as well as an explanation for the methodology that led to these results.

### V. Conclusion

This paper demonstrates the vast potential of the mMPU for energy-efficient cryptographic hash algorithms through a case study with the SHA-3 function. We propose a novel variable rotation algorithm and an efficient mapping that exploits the inherent parallelism of the mMPU towards high-throughput SHA-3 execution. This provides a massive throughput per watt of 1,422 Gbps/W, improving the state-of-the-art by 4.6×.

### ACKNOWLEDGMENT

This work was supported by the European Research Council through the European Union’s Horizon 2020 Research and Innovation Program under Grant 757259.

### REFERENCES

[1] A. Menezes, P. V. Oorschot, and S. Vanstone, “Handbook of applied cryptography (1st edition),” CRC Press, 1997.
[2] M. J. Dworkin, “SHA-3 standard: Permutation-based hash and extendable-output functions,” NIST, 2015.
[3] A. Akin et al., “Efficient hardware implementations of high throughput SHA-3 candidates Keccak Luffa and Blue Midight for single-and multi-message hashing,” in SIGCONF, 2010.
[4] P. Pessl and M. Hutter, “Pushing the limits of SHA-3 hardware implementations to fit on RFID,” CHES, 2013.
[5] M. M. Wong et al., “A new high throughput and area efficient SHA-3 implementation,” ISCAS, 2018.
[6] R. Ramanarayanan, “1GHz 50mW reconfigurable multi-mode SHA hashing accelerator in 45nm CMOS,” in ESSCIRC, 2010.
[7] J. Strömbergson, “Implementation of the Keccak hash function in FPGA devices,” Technical report, InformAsic AB, 2008.
[8] K. Nagarajan et al., “SHINE: A novel SHA-3 implementation using ReRAM-based in-memory computing,” ISLPED, 2019.
[9] D. Bhattacharjee, V. Pudi, and A. Chattopadhyay, “SHA-3 implementation using ReRAM based in-memory computing architecture,” 18th International Symposium on Quality Electronic Design (ISQED), 2017.
[10] C. Yang and Z. Chen, “A processing-in-memory implementation of SHA-3 using a voltage-gated spin hall-effect driven MTJ-based crossbar,” in GLSVLSI, 2019.
[11] N. Talati et al., “Logic design within memristive memories using memristor-aided logic (MAGIC),” TNAO, 2016.
[12] M. S. Q. Truong et al., “RACER: Bit-pipelined processing using resistive memory,” in MICRO, 2021.
[13] L. Shuangchen et al., “DRISA: A DRAM-based reconfigurable in-situ accelerator,” MICRO, 2017.
[14] S. Aga et al., “Compute caches,” HPCA, 2017.
[15] D. Reis et al., “Computing in memory with FeFETs,” in ISLPED, 2018.
[16] S. Kvatinsky, “Making real memristive processing-in-memory faster and reliable,” in CNNA, 2021.
[17] L. Chua, “Memristor-the missing circuit element,” TCT, 1971.
[18] J. Borghetti et al., “Memristive’ switches enable ‘stateful’ logic operations via material implication,” Nature, 2010.
[19] S. Kvatinsky et al., “MAGIC—memristor-aided logic,” TCAS-II, 2014.
[20] S. Gupta et al., “FELIX: Fast and energy-efficient logic in memory,” in ICCAD, 2018.
[21] J. Reuben et al., “Memristive logic: A framework for evaluation and comparison,” in PATMOS, 2017, pp. 1–8.
[22] G. Berton, “Keccak,” Advances in Cryptology – EUROCRYPT, 2013.
[23] O. Leitersdorf, R. Ronen, and S. Kvatinsky, “MatPIM: Accelerating matrix operations with memristive stateful logic,” in ISCAS, 2022.
[24] O. Leitersdorf et al., “AritPIM: High-throughput in-memory arithmetic,” in arXiv, 2022.
[25] R. Ronen et al., “The Bitlet Model: A parameterized analytical model to compare PIM and CPU systems,” JETC, 2021.
[26] B. Patmani, “Computer Arithmetic: Algorithms and hardware designs,” Oxford University Press, vol. 2, 2010.
[27] Available at https://github.com/BatelOved/HashPIM.