IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 24 January 2024, accepted 18 February 2024, date of publication 7 March 2024, date of current version 14 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3373311

RESEARCH ARTICLE

MATSA: An MRAM-Based Energy-Efficient Accelerator for Time Series Analysis

IVAN FERNANDEZ[1,2], CHRISTINA GIANNOULA[2,3], ADITYA MANGLIK[2], RICARDO QUISLANT[1], NIKA MANSOURI GHIASI[2], JUAN GÓMEZ-LUNA[2], (Member, IEEE), ELADIO GUTIERREZ[1], OSCAR PLATA[1], AND ONUR MUTLU[2], (Fellow, IEEE)

[1]Department of Computer Architecture, University of Málaga, 29071 Málaga, Spain
[2]Department of Information Technology and Electrical Engineering (D-ITET), ETH Zürich, 8092 Zürich, Switzerland
[3]School of Electrical and Computer Engineering, National Technical University of Athens, Zografou, 157 80 Athens, Greece

Corresponding author: Nika Mansouri Ghiasi (mnika@ethz.ch)

This work was supported in part by TIN2016-80920-R and UMA18-FEDERJA-197 Spanish Projects; in part by the High Performance, Edge And Cloud computing (HiPEAC) Collaboration Grants; in part by SAFARI Group's Industrial Partners, especially Advanced Semiconductor Materials Lithography (ASML), Facebook, Google, Huawei, Intel, Microsoft, and VMware; and in part by Semiconductor Research Corporation.

ABSTRACT *Time Series Analysis* (*TSA*) is a critical workload to extract valuable information from collections of sequential data, e.g., detecting anomalies in electrocardiograms. Subsequence Dynamic Time Warping (sDTW) is the state-of-the-art algorithm for high-accuracy TSA. We find that the performance and energy efficiency of sDTW on conventional CPU and GPU platforms are heavily burdened by the latency and energy overheads of data movement between the compute and the memory units. sDTW exhibits low arithmetic intensity and low data reuse on conventional platforms, stemming from poor amortization of the data movement overheads. To improve the performance and energy efficiency of the sDTW algorithm, we propose MATSA, the first Magnetoresistive RAM (MRAM)-based Accelerator for TSA. MATSA leverages Processing-Using-Memory (PUM) based on MRAM crossbars to minimize data movement overheads and exploit parallelism in sDTW. MATSA improves performance by 7.35×/6.15×/6.31× and energy efficiency by 11.29×/4.21×/2.65× over server-class CPU, GPU, and Processing-Near-Memory platforms, respectively.

INDEX TERMS Time series analysis, processing-using-memory, memory-bound, emerging technologies.

I. INTRODUCTION

In the era of Internet-Of-Things and Big Data, emerging applications operate on petabyte-scale datasets that are increasingly difficult to store and analyze. Small sensors and edge devices continuously generate data sampled over time, resulting in time-ordered observations (e.g., temperature or voltage). Such a collection of data values is referred to as a *time series* (TS) [1]. TS is a common data representation in many real-world scientific applications, including sensing, genomics, neuroscience, financial markets, epidemiology, and environmental sciences [2].

Time series analysis (TSA) splits the time series into *subsequences* of consecutive data points to extract valuable information from large datasets. This information can help filter relevant subsequences to minimize the cost of applying complex and expensive domain-specific analysis algorithms. A real-life example is the detection of anomalies in an electrocardiogram and the elimination of subsequences that indicate normal behavior [3]. TSA determines subsequences of interest using different similarity approaches, such as the Euclidean Distance (ED) or the subsequence Dynamic Time Warping (sDTW). Prior work demonstrates that sDTW provides a higher precision than ED in most scenarios [4]; as such, we focus on optimizing sDTW algorithm for TSA analysis.

sDTW is an embarrassingly parallel workload, because each query can be executed without data dependencies from other queries by multiple concurrent processing units. However, sDTW builds a 2D dynamic programming matrix that incurs quadratic runtime and memory complexity. To understand the bottlenecks of sDTW in state-of-the-art conventional CPU

The associate editor coordinating the review of this manuscript and approving it for publication was Harikrishnan Ramiah.

and GPU architectures, we comprehensively characterize the kernel's performance on these platforms (section II-D). We observe significant performance and energy efficiency overheads in sDTW due to: 1) underutilization of the execution units, and 2) a large number of expensive main memory accesses. The first problem stems from the low number of operations that the sDTW kernel executes per byte brought from memory, which keeps the arithmetic units idle for the largest part of the execution time. The second problem stems from the large memory footprint of the dynamic programming matrix, causing poor spatial and temporal locality. Consequently, sDTW exhibits poor performance on CPU and GPU platforms.

To overcome the memory access challenge, prior works [5], [6], [7] have considered *memory-centric* platforms that integrate processing and storage elements on the same chip to reduce data movement across the constrained data bus that connects a CPU to main memory [8], [9]. Based on that, we implement and characterize sDTW in a real Processing-Near-Memory (PNM) platform, *UPMEM* [10], and observe that this new platform does *not* provide performance benefits compared to CPU and GPU executions, due to the large latency of simple operations such as addition and comparison operators. Overall, we conclude that the sDTW kernel exhibits memory-bound behavior on CPU and GPU platforms and compute-bound behavior on the PNM platform (section II-D).

In contrast to PNM, *Processing-Using-Memory* (PUM) [7], [11], [12], [13], [14], [15] executes operations using the memory cells and sense amplifiers, completely eliminating the memory and compute dichotomy. PUM enables 1) performing computation *in the memory array*, since the memory units that store the data also execute the computation, and 2) exploiting a much larger amount of parallelism available in the memory microarchitectures (as high as the number of crossbar columns available, i.e., thousands) compared to conventional CPU and GPU systems. From the technology perspective, non-volatile memories (NVM) offer a promising substrate to implement PUM [16]. However, different NVM substrates exhibit varying latency, energy, and endurance characteristics, a key design constraint for different accelerators. Magnetoresistive RAM (MRAM)-based PUM substrates offer low read/write latencies, low energy per operation, and high endurance [17]. Considering these characteristics, in this paper, we explore MRAM as a potential NVM substrate to accelerate the sDTW kernel.

To this end, our goal in this work is to leverage MRAM-based PUM to *enable high-performance and energy-efficient sDTW execution for a wide range of applications*. We propose *MATSA*, the first MRAM-based Accelerator for TSA. MATSA derives its performance benefits from three key mechanisms. First, MATSA decomposes sDTW's computational kernel into simple bitwise boolean computations and executes them in the MRAM crossbar. This key idea significantly minimizes data movement overheads as it is performed where data resides. Second, we implement a novel data mapping that reduces the runtime memory footprint of sDTW from quadratic to linear based on four vectors. This key idea enables computing the complete 2D dynamic programming matrix on-the-fly without storing it. Third, MATSA integrates an effective computation scheme that overcomes the inter-cell computation dependencies of the matrix by 1) following an anti-diagonal approach and 2) exploiting pipelining to increase parallelism.

We evaluate MATSA's performance based on state-of-the-art latency and energy characteristics of MRAM devices [18], [19]. To do so, we implement an in-house simulator for MATSA and select 64 synthetic datasets to understand its design tradeoffs. Then, we use six real-world datasets (*Human*, *Song*, *Penguin*, *Seismology*, *Power* and *ECG*) to compare three different versions of MATSA against other state-of-the-art platforms, showcasing its applicability to a wide range of real case scenarios. Our evaluation shows that MATSA improves performance by 7.35×/6.15×/6.31× and energy efficiency by 11.29×/4.21×/2.65× over server-class CPU, GPU, and PNM platforms, respectively.

In summary, we make the following novel contributions:

- We thoroughly characterize the state-of-the-art sDTW time series analysis (TSA) algorithm's performance and energy efficiency on conventional CPU, GPU, and PNM (UPMEM) platforms. Our characterization leads to new observations about the characteristics of sDTW that limit its acceleration in current conventional hardware.
- We propose *MATSA*, the first MRAM-based Accelerator for TSA. MATSA 1) exploits a novel data mapping tailored for MRAM substrates that reduce memory footprint in sDTW, 2) efficiently performs computation in-memory to avoid off-chip data movement, and 3) provides an effective computation scheme to increase parallelism.
- We conduct a comprehensive evaluation of MATSA across a diverse set of synthetic and real-world datasets. Our results showcase 6.60× average improvement in overall performance and a average 6.05× boost in energy efficiency over state-of-the-art compute-centric and memory-centric platforms.

II. BACKGROUND AND MOTIVATION

A. TIME SERIES ANALYSIS

A *time series* T is a sequence of n data points t_i, where $1 \leq i \leq n$, collected over time. A subsequence of T, also known as a *window*, is denoted by $T_{i,m}$, where i is the index of the first data point, and m is the number of samples in the subsequence, with $1 \leq i$, and $m \leq n - i$.

There are two main approaches to perform time series analysis: 1) the self-join, and 2) the query-filtering. In self-join, all sequences of a given time series are compared against the remaining subsequences of the same time series. In contrast, query filtering compares a set of queries against a reference.

Time series analysis algorithms usually define a distance metric to measure the similarity between two subsequences. Based on such distance metric, the literature classifies the subsequences with low distance as *motifs* [20] (similarities) and high distance as *discords* [21] (anomalies). The state-of-

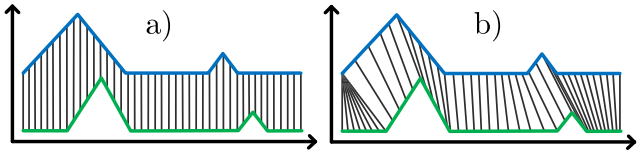

FIGURE 1. Example of similarity calculation between two subsequences (blue and green). The one-to-one approach in a) provides a low similarity as it only compares each i^{th} point of blue with each i^{th} point of green. In contrast, DTW in b) successfully matches the points of the subsequences.

the-art set of tools to perform time series analysis is Matrix Profile [22] (MP). Due to lower computation requirements, prior MP algorithms utilize one-to-one Euclidean Distance as the similarity metric. Recent proposals [4] have started to utilize Dynamic Time Warping (DTW)-based solutions because of higher precision [23]. DTW enables the detection of events of interest in out-of-sync subsequences, e.g., in subsequences that have different sampling rates.

Figure 1 shows the key difference between the one-to-one and the DTW approaches, in which we compare two similar-shape subsequences that differ in their offset and scale.

We observe that the DTW algorithm offers better results as it compares a given point with respect to several potential candidates (i.e., determines the best alignment). In contrast, one-to-one executes point-to-point alignment that cannot determine the best alignment in the presence of an offset. One-to-one can be considered as a special case of DTW where the *warping window* is set to '1'. Therefore, we aim to optimize DTW, a more generic and high-precision algorithm, to provide a TSA accelerator for a wide range of applications.

B. TIME SERIES ANALYSIS APPLICATIONS

Time series analysis constitutes one of the most important and general data mining primitives for a wide range of real-world applications [24]: epidemiology, genomics, neuroscience, medicine, environmental sciences, economics, and many more. Table 1 presents a few examples for applications of TSA.

TABLE 1. Time series analysis main applications.

Field	Reference	Field	Reference
Bioinformatics	[25]	Speech Recognition	[26]
Robotics	[27]	Weather Prediction	[28]
Neuroscience	[25]	Entomology	[29]
Machine Learning	[30]	Geophysics	[31]
Econometrics	[32]	Statistics	[33]
Finance	[34]	Control Engineering	[35]
Signal Processing	[36]	Pattern Recognition	[37]
Communication	[38]	Medicine	[39]
Astronomy	[40]	Social Networks	[41]
Clustering	[30]	Classification	[42]
Earthquakes	[43]	GPS Tracking	[44]
Virtual Reality	[45]	Gesture Recognition	[46]
Trajectories	[47]	Traffic Monitoring	[48]

In statistics, econometrics, meteorology, and geophysics, the primary goal of time series analysis is prediction and forecasting. At the same time, in signal processing, control engineering, and communication engineering, it is used for signal detection and estimation. In data mining, pattern recognition, and machine learning, time series motif and discord discovery are used for clustering, classification, anomaly detection, and forecasting. Finally, the most important application of time series motif and discord discovery is clustering seismic data and discovering earthquake pattern clusters from the continuous seismic recording. Consequently, seismic clustering can be applied to earthquake relocation and volcano monitoring to help improve earthquake and volcanic hazard assessments.

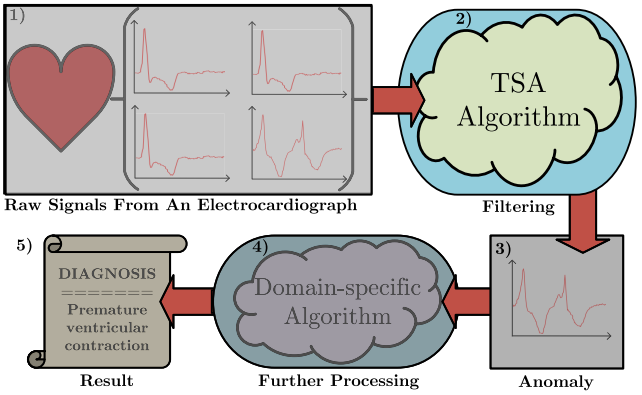

FIGURE 2. Example TSA application, where TSA acts as a filter to avoid most of the computation. TSA selects the relevant queries (anomalies) and discards the irrelevant ones.

Within this field, the subsequence Dynamic Time Warping (sDTW) algorithm is a fundamental kernel due to its superior accuracy and generality when compared to other TSA methods [4]. Examples of real-life use cases that can benefit from high-performance and energy-efficient sDTW are:

- **Circulatory Failure Detection in Intensive Care Units**. TSA consumes 90% of the end-to-end execution time [49]. Figure 2 describes the aforementioned process based on an example processing flow.
- **Electroencephalography (ECG)**. TSA is deployed to monitor and filter ECG readings when monitoring patients [50].
- **Earthquake Detection**. TSA is critical to process seismograph data and detect anomalies for further analysis [43].

C. DYNAMIC TIME WARPING (DTW)

DTW algorithm was first introduced by [51]. The first step of DTW is to compute the distance between a particular point from a subsequence and a set of points from another subsequence, only keeping the minimum of them. This process is repeated for all the points of the first subsequence. Then, DTW computes the addition of all distances, providing a similarity measure between the subsequences (the lower the distance, the higher the similarity).

Assuming that we have two subsequences, Q (query) and R (reference), of length n and m, respectively, where:

$$Q = q_1, q_2, \ldots, q_i, \ldots, q_n \quad R = r_1, r_2, \ldots, r_j, \ldots, r_m \quad (1)$$

DTW constructs an n-by-m scoring matrix (S) to determine the similarity between the two subsequences. Each (i^{th}, j^{th}) cell of the matrix ($s_{i,j}$) is filled in two steps. First, the algorithm calculates the distance $d(q_i, r_j)$ between the two corresponding points of the subsequences. There are several approaches to calculate such distance, while $d(q_i, c_j) = abs(q_i - c_j)$ and $d(q_i, c_j) = (q_i - c_j)^2$ are the most common ones. Second, the distance value is added to the minimum of the three neighboring cells as follows:

$$s_{i,j} = d(q_i, c_j) + min(s_{i-1,j-1}, s_{i-1,j}, s_{i,j-1}) \quad (2)$$

The algorithm fills the entire matrix using dynamic programming. Then, the goal is to find the best alignment (i.e., minimum accumulated cost), known as the *warping path* (W). W is a contiguous set of matrix cells that defines the best mapping between Q and R.

Subsequence Dynamic Time Warping (sDTW): sDTW is a more general DTW algorithm that allows the query to be aligned with part of the reference. Algorithm 1 presents the pseudocode of sDTW.

Algorithm 1 Subsequence DTW (sDTW)

```
1: procedure sDTW(Q,R)
2:     S ← zeros(N, M);
3:     S[0, 0] = dist(Q[0], R[0]);
4:     for i ← 1 to N do
5:         S[i, 0] ← S[i − 1, 0] + dist(Q[i], R[0]);
6:     for i ← 1 to N do
7:         for j ← 1 to M do
8:             S[i, j] ← dist(Q[i], R[j]) +
9:             min(S[i − 1, j − 1], S[i, j − 1], S[i − 1, j]);
       return min(S[N, :])
```

First, sDTW initializes the matrix S with zeros. Second, it calculates the distance value of the top-left corner and then the remaining elements of the first row, taking into account the previous values. Third, it fills the remaining elements of the matrix using dynamic programming row by row. Finally, it returns the minimum element of the last row of the S matrix, which indicates the similarity between the query and the best alignment with (part of) the reference. The nested `for` loops (lines 6 and 7 in Algorithm 1) are responsible for the quadratic runtime and memory complexities.

D. BOTTLENECKS OF SDTW IN CONVENTIONAL AND PNM PLATFORMS

sDTW's quadratic computational complexity is challenging to overcome, especially when accurate results are required and algorithmic optimizations are insufficient. To determine the bottlenecks in conventional platforms, we perform a detailed characterization of parallelized and optimized sDTW kernels on CPU, GPU, FPGA, and PNM platforms.

1) CPU

We profile the performance of sDTW on an Intel Xeon Phi 7210 CPU using the Intel Advisor tool. We build the roofline plot and present the result in Figure 3-left. First, we observe that sDTW-CPU can utilize only 41% of the system's integer peak performance, i.e., 59 GINTOPS out of 145 GINTOPS, and exhibits low arithmetic intensity (0.55 INTOP/Byte). Second, the total memory traffic generated during runtime is 267 GB. In contrast, the memory footprint of the sDTW kernel is only 570 MB. This demonstrates that sDTW is a memory-bound kernel for CPU targets.

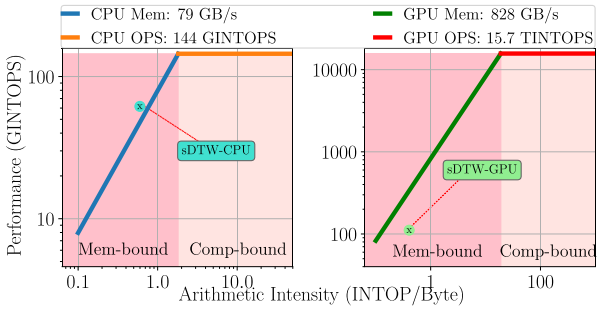

FIGURE 3. Roofline plots for sDTW on a many-core CPU platform (left) and a server-class GPU (right).

2) GPU

Several prior works propose accelerating sDTW using GPUs (e.g., [52]). However, these implementations are tailored and optimized for specific workload sizes. They rely on high-latency global memory when working with arbitrary-sized datasets, which results in large performance penalties compared to the optimal input size. To quantify the bottlenecks, we develop an optimized CUDA-based implementation that supports arbitrary subsequence sizes and characterize it on the NVIDIA Tesla V100 GPU. We analyze the sDTW kernel using NVIDIA Visual Profiler [53] and generate the roofline plot in Figure 3-right. We observe that sDTW-GPU's performance improves with respect to sDTW-CPU but utilizes merely 1% of the GPU's available peak performance. We explain this observation by 1) the low arithmetic intensity of sDTW and 2) the limited per-thread available local memory. Even increasing the available local memory does not improve performance and the algorithm hits the memory roof due to 1), thus greatly underutilizing the platform. Based on this analysis, we conclude that GPU is not a good target for sDTW kernels executing on arbitrary subsequence sizes, which is the common case in many applications.

3) FPGA

sDTW acceleration using FPGAs requires large onboard memory to achieve high performance. As most of the prior work based on FPGAs does not provide high on-chip memory capacity, data is distributed over the chip. We develop an optimized FPGA implementation targeting a Xilinx Alveo U50 and build the roofline model in Figure 4-left. We observe that the eight compute units that fit in the FPGA achieve less

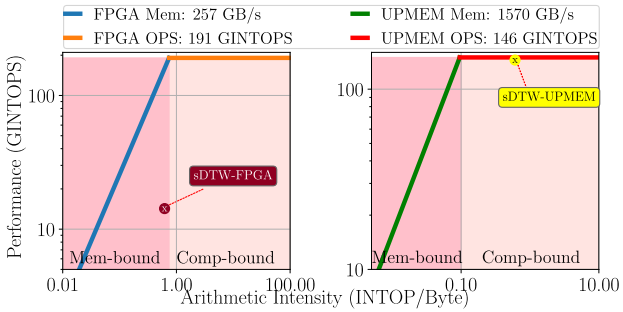

FIGURE 4. Roofline plots for sDTW on FPGA (left) and UPMEM (right) platforms.

than 7% of the available peak throughput and are insufficient to exploit the inherent parallelism in the sDTW kernel.

> **Key Observation 1:** Conventional architectures fail to provide a high performance and energy efficient acceleration solution because execution time and energy are wasted on the data movement between memory and processing units.

4) PROCESSING-NEAR-MEMORY (PNM)

PNM platforms place processing units in the same die as memory units. The idea behind this paradigm is to exploit the lower latency and higher bandwidth available in memory and mitigate the data movement overheads between the processing units and memory. To evaluate the performance and energy efficiency of sDTW on PNM, we implement an optimized version of the algorithm on UPMEM [54], the first commercially available server-class PNM platform. We build the roofline model in Figure 4-right and observe that sDTW is compute-bound in UPMEM. This observation can be attributed to the low-power general-purpose cores in UPMEM that offer poor throughput (146 GINTOPS in contrast to 15700 GINTOPS for the GPU). As arithmetic operations are at the core of sDTW, PNM cannot provide high performance for it. We also observe that UPMEM reduces the energy consumption by 37% with respect to the GPU by reducing the data movement overheads (section IV-C). However, poor performance in contrast to the GPU inhibits the effective usability of the platform for the sDTW kernel.

> **Key Observation 2:** General-purpose PNM substrates provide higher energy efficiency compared to CPU/GPU/FPGA platforms. However, they fail to offer a high performance solution because of the limited arithmetic computation throughput supported by the hardware.

E. OVERCOMING BOTTLENECKS IN TSA

1) NEED FOR PROCESSING-USING-MEMORY (PUM)

We observe that when executing the sDTW kernel, 1) CPU, GPU, and FPGA platforms are memory-bound, and 2) PNM platforms are compute-bound. In contrast to these platforms, PUM accelerators execute operations directly using the memory cells where data resides [15]. PUM enables 1) exploiting large internal memory bandwidth for memory-bound kernels, and 2) exploiting massive computation parallelism (as high as each bitline) for compute-bound kernels, overcoming key restrictions of CPU, GPU, FPGA and PNM architectures. Based on these observations, *we argue that an accelerator based on PUM is needed to improve TSA's performance and energy efficiency providing a balanced solution*.

2) CELL TECHNOLOGY CHOICE

A PUM-based accelerator's performance, energy efficiency, and endurance depend on the underlying substrate's cell technology; thus, it is a critical design choice. Non-Volatile-Memories (NVM) offer a low-energy substrate for PUM as they do not require periodic refresh operations in contrast to DRAM-based PUM [55]. However, it is challenging to support frequent write operations as NVM-based PUM architectures due to significant write latency and low endurance [56]. Table 2 presents the characteristics of NVM technologies we considered for accelerating the sDTW kernel. We discard NAND Flash, ReRAM, and PCM in the first step due to their low endurance and high write latency. Next, we consider FRAM due to its high endurance but discard it due to the high read latency. We then consider MRAM technologies (section II-F) and discard STT-MRAM due to a high write latency. In contrast to STT-MRAM, SOT-MRAM offers 1) high endurance, 2) low read and write latencies, and 3) CMOS compatibility that eases manufacturability. Considering these characteristics, we argue that SOT-MRAM is a promising substrate for implementing PUM accelerators for kernels with frequent write operations, and evaluate its feasibility for accelerating the sDTW kernel.

TABLE 2. Characteristics of different NVM technologies [57].

Technology	Write/Read Energy	Write/Read Time	Write Cycles
NAND Flash	470pJ / 46pJ	200μs / 25.2μs	10^5
ReRAM	1.1nJ / 525fJ	10μs / 5ns	10^5
PCM	13.5pJ / 2pJ	150ns / 48ns	10^7
FRAM	1.4nJ / 1.4nJ	120ns / 120ns	10^{15}
STT-MRAM	2nJ / 34pJ	250ns / 10ns	$> 10^{15}$
SOT-MRAM	334pJ / 247pJ	**1.4ns / 1.1ns**	$> 10^{15}$

We conclude that the MRAM-PUM acceleration approach has the potential to overcome TSA's bottlenecks and provide a faster and more efficient solution than the state-of-the-art.

F. MRAM-BASED PUM COMPUTATION

Many prior works demonstrate significant performance and energy efficiency improvements for machine learning workloads via PUM in resistive crossbars [58] by exploiting matrix-vector multiplication. Other approaches can exploit bitwise operations with high performance and energy savings [59], [60], [61]. Figure 5-a shows a typical crossbar

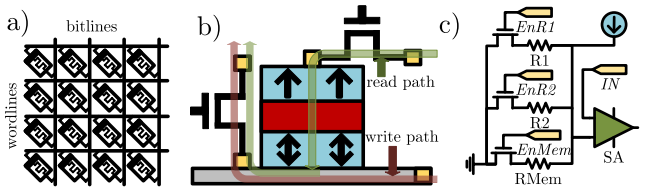

FIGURE 5. a) Crossbar organization. b) Magneto-resistive cell. c) Reconfigurable SA that performs in-memory operations based on the voltage variations across the bitline.

organization with memory cells connected using bitlines and wordlines.

Figure 5-b shows the basic structure of a Spin-Orbit Torque (SOT)-MRAM cell composed of a stack of Magnetic Tunnel Junctions (MTJs) (cyan and red blocks in the figure) and a Heavy Metal Layer (grey block in the figure).

- **Magnetic Tunnel Junction (MTJ)**. Consists of a fixed layer with a pinned magnetization direction, a free layer whose magnetization can be changed, and an insulating tunnel barrier between them.
- **Heavy Metal Layer**. This layer is placed next to the MJT to facilitate the spin-orbit torque effect. Common heavy metals used include tantalum (Ta) and tungsten (W).

The change of orientation of one of the layers of the stack results in a variation in the device's electrical resistance. However, compared to Spin-Transfer-Torque MTJ (STT-MTJ) [57], SOT-MJT features separated read and write paths, enhancing endurance and widening the read/write margin. Then, sense amplifiers interpret the resulting voltage as boolean:

- **Read Operation**. During a read operation, the resistance of the MTJ is measured. The resistance is sensitive to the relative alignment of the magnetization in the fixed and free layers, allowing the stored data (Boolean values representing 0 or 1) to be read.
- **Write Operation**. During a write operation, an electric current is applied through the heavy metal layer, inducing a spin current. This spin current exerts torque on the free layer, causing its magnetization direction to switch and changing the stored Boolean data.

Unlike STT-MTJ, which faces read disturbance issues limiting the read circuit frequency, SOT-MTJ allows for flexible adjustment of current magnitude in the read circuit without concerns about read disturbance effects. As a consequence, it enables more accurate sensing which is crucial to implement in-memory operations. This suggests SOT-MRAM as a better candidate for PUM applications.

Bitwise PUM Mechanism: The matrix-vector PUM mapping proposed in prior works cannot be applied to dynamic programming (DP) algorithms (e.g., sDTW) since they perform matrix-vector multiplication. DP requires computing a 2D scoring matrix by traversing it row-by-row. Moreover, prior crossbar substrates offer limited support for other operations (e.g., minimum calculation). To overcome this challenge, MAGIC [62] proposes decomposing complex operations into simple Boolean functions (e.g., AND, NOR, XOR) to support them in the substrate. The key idea is to vertically map the operands (e.g., 32-bit integers) to the crossbars' columns using (typically) one bit per cell (e.g., each operand value takes 32 bits of a given column). Then, the desired operation (e.g., addition) is decomposed to simple bitwise operations (e.g., NOR) and performed bit-by-bit via sequentially activating two cells for each operand simultaneously. This approach creates a difference in the voltage over the bitline depending on the content of the activated cells, which depends on the resistance they hold. Then, a modified sense amplifier calculates the result based on that voltage difference and thresholds, storing it in a cell of the same column. While this process is inherently sequential and the latency per operation is higher than a CMOS-based approach, the 1) independence across columns and 2) the lack of data movement enables immense parallelism and, thus, an overall higher throughput than CMOS-based solutions. Figure 5-c shows a sense amplifier (SA) slightly modified with respect to commodity ones, including different voltage thresholds for the operations.

III. MATSA ARCHITECTURE

A. OVERVIEW

MATSA is an MRAM-based Accelerator for Time Series Analysis. Figure 6 presents an overview of our proposed architecture. MATSA is composed of several chips divided into multiple *banks*. Banks belonging to the same chip share buffers and I/O interfaces and work in a lock-step approach. Each bank is composed of several *Multiple Memory Matrices (MATs)*. The MATs share a *Global Row Buffer (GRB)* and are connected to a *Global Row Decoder (GRD)*. We place a *Local Row Buffer (LRB)* for every pair of subarrays to improve performance. Each subarray is composed of magnetoresistive devices that are connected to the *Write Word Lines* (WWL), *Write Bit Lines* (WBL), *Read Word Lines* (RWL), *Read Bit Lines* (RBL), and *Source Lines* (SL). The compute-enabled subarrays perform the sDTW computation using Reconfigurable Sense Amplifiers (RSAs).

The execution flow is orchestrated by a hierarchy of small controllers implemented as finite state machines (FSMs). MATSA comprises of 1) a global controller that orchestrates inter-bank flow, 2) inter-mat controllers that take care of the inter-mat flow, and 3) subarray controllers that activate the memory rows and drive the RSAs to run sDTW's algorithm.

B. MATSA SUBARRAYS

MATSA subarrays are comprised of MRAM cells following a crossbar organization and can work either in regular memory or compute mode. This is a desirable feature since our design consists of 1) subarrays that temporarily buffer the data until they are being processed and 2) subarrays that perform the actual computation. Adjacent subarrays are connected using pass gates and aux columns (purple one in Figure 6) to enable the data flow through the hierarchy.

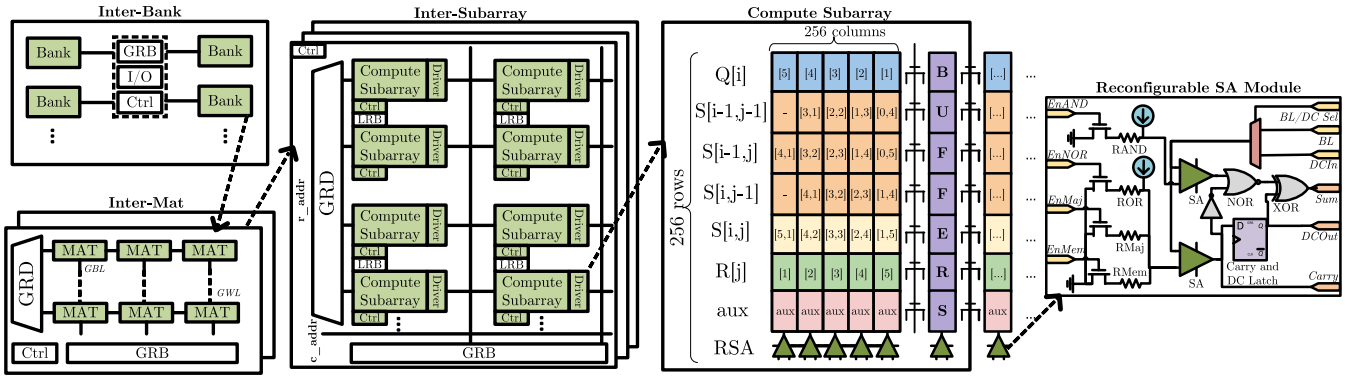

FIGURE 6. MATSA's high-level architecture and data mapping flow.

1) MEMORY SUBARRAYS

MATSA subarrays in *regular memory mode* support both read and write data operations and work in the same way as conventional non-PUM-enabled memory.

2) COMPUTE SUBARRAYS

MATSA subarrays working in *compute mode* perform bit-wise operations on input data located in cells of the same column. This enables the parallel execution of many operations since all columns in the subarray work in parallel. The key idea is to select two or three input values simultaneously using the Memory Row Decoder (MRD). This produces an equivalent resistance that depends on the content of the selected cells and modifies the sensing voltage across the column accordingly. MATSA's Ctrl can select different operations from the Reconfigurable Sense Amplifiers (RSAs) that are placed per column. We modify the RSAs to execute operations by equipping them with different resistances to model the voltage thresholds, logic gates (i.e., NOR, XOR, INV), a register, and a multiplexer (see Figure 6). The RSAs in Compute subarrays support the same operations as memory subarray RSAs, enabling switching between operating in compute and memory modes.

C. PUM OPERATIONS

MATSA implements the following PUM operations to support the execution of sDTW (detailed in Algorithm 1):

- **Vertical Row Copy**. MATSA executes consecutive memory read and write operations in the same cycle to improve performance by activating two rows simultaneously. In the first half cycle, the subarray's MRD activates the source row read by the LRB. Next, the destination row is activated to store the data in the second half cycle. This mechanism works at MAT and bank levels using the Global Row Buffer (GRB) to accelerate the copies across the hierarchy.
- **Diagonal Row Copy**. The Ctrl executes a diagonal copy shift data between adjacent columns. The Ctrl leverages the available registers in the RSA and the interconnections between the RSAs. The operation is executed in two steps. First, the RSA reads the value in the source column. Second, the destination RSA (in an adjacent column) reads the value from the source RSA and writes it to its column.
- **Addition/Subtraction**. MATSA executes Bit-serial addition/subtraction across columns. The Ctrl executes operations starting from the least significant bit of the two operands until the most significant bit. Every bit operation requires two memory cycles, further divided into four half cycles. In the first half-cycle, the RSAs read voltage difference across all cells activated in the same bit lines as input operands and calculate the *Sum*. The RSA updates the Sum based on the stored Carry value in the register. In the second half-cycle, the RSAs write the *Sum* value to the destination cell. In the third half-cycle, the RSAs calculate the new *Carry* value based on a majority function of the operand rows and an auxiliary row reserved for the Carry bit. In the fourth half-cycle, RSAs write the new Carry value in the auxiliary row for the next Carry calculation.
- **Absolute Calculation**. To calculate the absolute value, MATSA first checks the sign bit, leading to two possible scenarios: 1) if the number is positive, no change is needed; otherwise, 2) if the number is negative, MATSA inverts the bits of the number and adds '1' to the result (similar to 2's complement).
- **Minimum Value**. To calculate the minimum value between three elements, MATSA performs two comparisons based on the subtraction operation. First, it calculates the difference between the two numbers. Second, it checks the resulting sign from the previous step and selects one of the two numbers for comparison against the third. The final comparison sign determines the minimum between three values. The logic can be similarly extended for comparing more than three values.

D. DATA MAPPING

Section II-D demonstrates that sDTW is an embarrassingly parallel algorithm. We design MATSA's data mapping to leverage MRAM's parallel column-wise computation

capability. Three data structures are involved in the sDTW computation: 1) reference sequence (of length O(M)), 2) query sequence (of length O(N)), and 3) the warping matrix (dynamic programming matric with size O(NM)). The data structures are mapped to the subarray as follows:

- **Reference Elements (`R[j]`)**. We vertically map each reference element to 32 cells of a column. If 1) the number of available columns is larger than the number of elements in reference, we replicate the reference to multiple columns to increase parallelism (distributing the queries between them). If 2) the number of available columns is lower than the number of elements in reference, we divide the query and complete the process in sequential batches. No action is needed if 3) available columns are equal to the number of elements in reference.
- **Query Elements (`Q[i]`)**. We vertically map each query element to 32 cells of a column. New query elements are introduced on the left side of the crossbar, and they are right-shifted in each successive step (see section III-E).
- **Current S_vector (`S[i, j]`)**. We define the current vector of the warping matrix as the S_vector. We vertically map each element of the S_vector to 32 cells of a column, being aligned with the query and reference elements (i and j indexes, respectively).
- **Temporal S_vectors (`S[i-1, j-1]`, `S[i-1, j]`, `S[i, j-1]`)**. We vertically map the three temporal vectors along the reference and query elements. Mapping the temporal vectors in the same subarray leverages parallelism in the subarray as each column can compute lines 8-9 of Algorithm 1 completely in parallel. Then, those vectors are efficiently updated also in parallel for the next iteration of the loop thanks to the vertical and diagonal row copies.
- **Aux Cells**. Each column has a slice of 64 cells used to hold the partial results during the execution flow.

We calculate the distance between each data point in the reference and the query by iterating over the current S_vector of the warping matrix (see Algorithm 1). Each element in the S_vector (mapped across different crossbar columns) requires accessing previous S_vector values that are mapped to the same column (i.e., $S[i-1, j]$) and adjacent columns (i.e., $S[i, j-1]$, $S[i-1, j-1]$). To break this data dependency, we add three temporal S_vectors in the crossbar array that are updated in each step of the computation: $S[i-1, j-1]$, $S[i-1, j]$ and $S[i, j-i]$ (see Figure 6). Overall, our optimization reduces the memory footprint from O(NM) (whole matrix) to O(4M) (S_vector plus three aux ones).

E. EXECUTION FLOW

MATSA's execution flow follows a wavefront approach, which reflects the computation pattern in dynamic programming applications. The motivation is that sDTW's matrix has to be computed in the wavefront manner due to inter-cell dependencies. Figure 7 shows an example of how we tackle this restriction by assuming one reference time series (red one) and two queries (green and ocher).

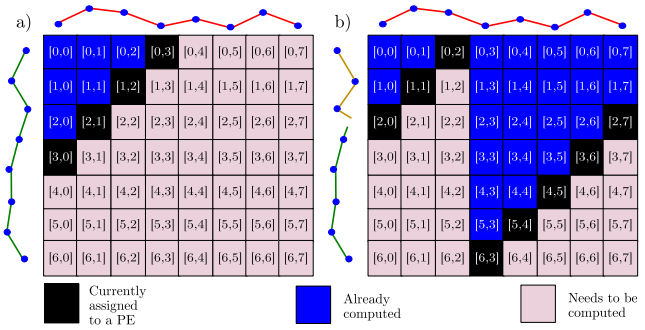

FIGURE 7. Wavefront-based sDTW computation in MATSA.

The key idea is to make computation flow diagonally by assigning one element in the wavefront to each processing element (PE), and using the *diagonal row copy* operation (section III-C) to shift data between columns on the wavefront. This is needed since each cell requires taking values from its left column, thus their data values need to be available prior to computation. Because of that, each PE advances computation in the vertical direction with one cell delay with its left PE, ensuring that the data needed to calculate the next value is available. Figure 7-a shows an initial state where the computation just started. In this example, only PEs where their column contain black rectangles are are performing computation. Note that in every step the wavefront introduces a new PE to the active set, achieving maximum performance after *number_of_PEs* steps. When reaching point, all PEs are able to perform useful work in a given execution step. Figure 7-b shows how this initialization phase can be amortized by pipelining. By introducing a new query to compare against the reference before the prior one finishes, MATSA ensures that all PEs have work to do even during the transitions between queries. Overall, this execution flow enables 1) leveraging the subarray columns in parallel for the query, and 2) creation of an inter-subarray pipeline to leverage parallelism across queries, i.e., by processing queries in parallel. The execution flow of each cell goes through the following steps:

1) **Distance Calculation**. Calculation of $dist(Q[i], R[j])$, which provides the first partial result $P1$. This process implies several substeps depending on the selected distance metric, (e.g., subtraction → absolute value).
2) **Minimum**. Calculation without storing the result of $min(S[i-1, j-1], S[i-1, j], S[i, j-1])$, which produces the value for the next step $S1$.
3) **Addition**. Calculation of the addition between the minimum value selected in the previous step ($S1$) and the partial result $P1$.
4) **Diagonal Copy**. Copying the $S[i, j]$ vector into the $S[i, j-1]$ vector shifted by one to the right.
5) **Diagonal Copy**. Copying the $S[i-1, j]$ vector into the $S[i-1, j-1]$ vector shifted by one to the right.
6) **Vertical Copy**. Copying the $S[i, j]$ vector into the $S[i-1, j]$ vector.

```
void matsa(DTYPE * ref, DTYPE * queries,  uint64_t *
    ref_size, uint64_t * query_sizes, uint64_t n_queries,
     char * mode, char * dist_metric, DTYPE anomaly_thres
    , bool * anomalies, DTYPE * distances)
```

LISTING 1. MATSA's host interface function.

7) **Diagonal Copy**. Copying the $Q[i]$ vector into the same $Q[i]$ vector but shifted one position to the right.

F. PROGRAMMING INTERFACE AND SYSTEM INTEGRATION

1) PROGRAMMING INTERFACE

We expose an API (Listing 1) that allows to invoke MATSA from the host processing unit.

MATSA expects input data to be in a supported type-/precision DTYPE (integer: `int8`, `int16`, `int32` or `int64`; fixed-point: `fp32` or `fp64`), the selected mode (either *query_filtering*, where queries are compared against the reference or *self_join*, where slices of the reference are compared against themselves) and the distance metric (*abs_diff* or *square_diff*). MATSA can also take an anomaly threshold, which returns an array with the detected ones.

2) SYSTEM INTEGRATION

MATSA is designed to work synergistically with the CPU to accelerate TSA. We propose three MATSA versions to meet the requirements of different environments, as we describe next.

a) **MATSA-HPC**. A high-performance PCIe-based accelerator intended to be integrated into servers.
b) **MATSA-Embedded**. A small chip intended to be integrated with edge devices (e.g., sensors).
c) **MATSA-Portable**. A USB-based accelerator intended for use in desktops and laptop computers.

IV. EVALUATION

A. METHODOLOGY

To comprehensively quantify the performance and energy efficiency improvements of MATSA, we compare it with the following systems.

- **CPU-ARM (`cpuarm`):** 4-core ARM CPU @ 2.5GHz, 32KB L1 and 8GB LPDDR4.
- **CPU-i7 (`cpui7`):** 6-core (12 threads) Intel i7 × 86 CPU @ 3.2GHz, 64KB L1, 256KB L2, 12MB L3 and 64GB DDR4.
- **CPU-Xeon (`cpuxeon`):** Two 18-core (36 threads) Intel Xeon Gold 6154 × 86 CPUs @ 3GHz, 32KB L1, 1MB L2, 24.75 MB L3 and 768GB DDR4.
- **GPU (`gpu`)**: NVIDIA Tesla V100 with 32GB of HBM.
- **FPGA (`fpga`)**: Xilinx Alveo U50 with 8GB HBM memory.
- **UPMEM (`upmem`):** Server-class Processing-Near-Memory DIMMs with 2560 DPUs running at 425MHz [10].
- **MATSA-Embedded (`matsa-embedded`)**: consisting of 128 compute-enabled crossbars (1MB) and 896 regular-memory crossbars (7MB).

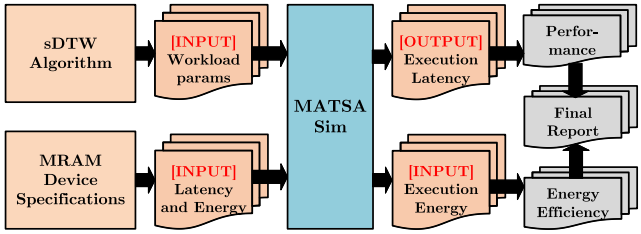

FIGURE 8. Overview of MATSA simulator.

- **MATSA-Portable (`matsa-portable`)**: consisting of 1024 compute-enabled crossbars (8MB) and 7168 regular-memory crossbars (56MB).
- **MATSA-HPC (`matsa-hpc`)**: consisting of 4096 compute-enabled crossbars (32MB) and 28672 regular-memory crossbars (224MB).

1) BASELINES

We use ZSim+Ramulator [63] and McPAT for the `cpuarm` platform. For the `cpui7` and `cpuxeon` platforms, we have access to the target hardware and measure performance and energy consumption values by averaging five repeated executions. The energy consumption is determined using Intel RAPL tools. To evaluate the performance of the `upmem` platform, we implement and optimize the sDTW algorithm as shown in Algorithm 1. To evaluate the performance on the `fgpa` platform, we implement the sDTW algorithm using High-Level Synthesis vendor tools from Xilinx and optimize the implementation to utilize eight compute units and maximize the utilization of the available HBM bandwidth. We evaluate the performance of the `gpu` platform by optimizing a CUDA-based implementation of sDTW to maximize the HBM bandwidth utilization via memory coalescing. We measure the GPU's energy consumption using the *NVIDIA-smi* tool.

2) MATSA

Due to the lack of a cycle-accurate simulator for MRAM-based accelerators, we implement an in-house simulator for MRAM-based PUM. Figure 8 shows an overview this simulator. We provide the workload characteristics and the MRAM device characteristics under study, and the simulator computes the performance and energy efficiency in return. We plan to release this simulator for public use of the community after acceptance of this work.

We perform a sensitivity analysis by sweeping MRAM devices' latency and energy from conservative to optimistic values based on MRAM device trends [64] listed in Table 3. Based on that, we conservatively select an operating point (highlighted in bold) for the evaluations taking into account realistic MRAM device progress projections. We input the workload parameters and MRAM characteristics obtained from the parameter sweep to the simulator to get the workload's execution time and energy consumption.

TABLE 3. MATSA design space exploration parameters.

Parameter	Values
Crossbar Size (cells)	256x256
Number of Crossbars	128, 256, 512, 1024, 2048, 4096
Read Latency (ns)	1, 3, **5**, 10, 20
Write Latency (ns)	1, 3, 5, **10**, 20
Read Energy (pJ)	20, **50**, 100
Write Energy (pJ)	30, **70**, 400

3) DATASETS

We perform MATSA's design exploration using the datasets in Table 4, which ease understanding of the tradeoffs. Then, we compare MATSA against baselines in real scenarios using the real datasets in Table 5. The data type for these evaluations is `int32`, which covers the data ranges of all the evaluated the workloads.

TABLE 4. Workloads used in MATSA characterization.

Parameter	Values
Reference Size	64K, 128K, 256K, 512K
Query Size	4K, 8K, 16K, 32K
Number of Queries	4K, 8K, 16K, 64K

TABLE 5. Real-world workloads used in our evaluation.

Time Series	Reference Size	Query Size	Num. Queries
Human	7997	120	128K
Song	20234	200	64K
Penguin	109842	800	32K
Seismology	1727990	64	16K
Power	1754985	1536	16K
ECG	1800000	512	16K

B. MATSA CHARACTERIZATION

We perform a design space exploration of MATSA taking into consideration performance parameters of the cells (i.e., read/write latencies and energies).

1) READ/WRITE LATENCIES

We evaluate how changing the read/write latencies affects the execution time and present the results in Figure 9. We observe that, increasing read latency by 10× incurs a 4.7× execution time penalty, while increasing the write latency incurs a 6.5× penalty.

> **Key Observation 3:** using a low write latency memory technology is crucial for MATSA's design.

2) READ/WRITE ENERGIES

We evaluate how the total execution energy varies with the per word write/read energy, and show the results in Figure 10. We observe here that the contributions of read energy and write energy are similar, thus both of them have to be carefully taken into consideration.

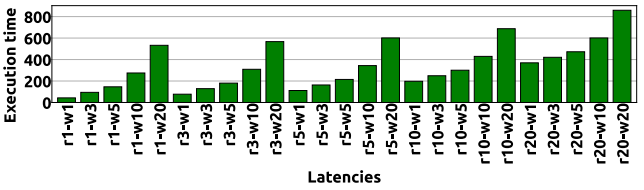

FIGURE 9. Execution time when varying cell read and write latencies (ref_size = 128K, query_size = 8K, num_queries = 8K, matsa_cols = 128K).

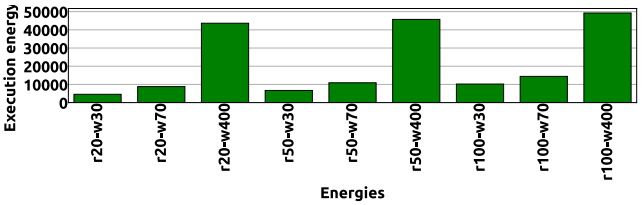

FIGURE 10. Execution energy when varying cell read and write energies (ref_size = 128K, query_size = 8K, num_queries = 8K, matsa_cols = 128K).

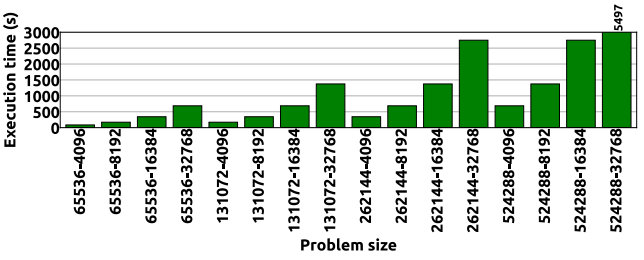

FIGURE 11. Execution time when varying dataset sizes (num_queries = 8K, matsa_cols = 128K).

> **Key Observation 4:** read energy contributes 45% and write energy contributes 55% to the total energy consumption of a given execution.

3) DATASET SIZES

First, we evaluate how the execution time varies with different dataset sizes (i.e., ref_size and query_size) and present the results in Figure 11. Second, we evaluate how the execution energy varies with different dataset sizes and present the results in Figure 12. We observe that both reference size and query size contribute equally to the execution time and energy. This happens because the total number of operations needed is directly proportional to ref_size×query_size. Our observation corroborates our earlier analysis stating that query-specific sDTW implementations do not fairly represent GPU performance, and there is a need for a more general solution.

> **Key Observation 5:** Total execution time and energy consumption are proportional to both ref_size and the query_size.

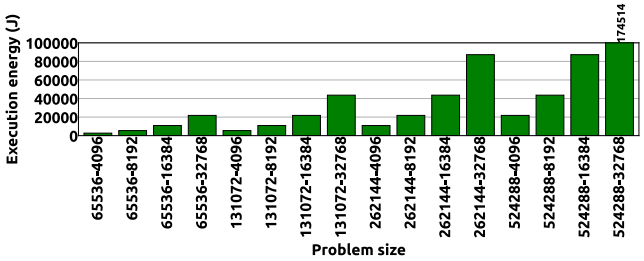

FIGURE 12. Execution energy when varying dataset sizes (num_queries = 8K, matsa_cols = 128K).

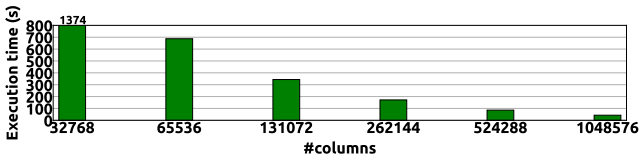

FIGURE 13. Execution time when varying MATSA sizes.

4) MATSA SIZES

We evaluate how the execution time varies when changing the number of MATSA's compute-enabled columns in Figure 13. MATSA provides almost-ideal scaling.

> **Key Observation 6:** Bit-serial computation across columns enables almost-ideal scaling when increasing the size of the workload.

5) ENDURANCE

Assuming that MATSA is built using 5/10ns rd/wr cells and runs 24/7 for ten years, we estimate that each cell will be written $\approx 4 \times 10^9$ times. Based on Table 2, limited-endurance cells (e.g., ReRAM) would fail within one day. In contrast, high-endurance cells (10^{15} writes for SOT-MRAM) can provide a very large usable lifetime.

6) HARDWARE OVERHEADS

MATSA introduces hardware overheads in two components: 1) Reconfigurable SAs and 2) MATSA controllers. Reconfigurable SAs add 13 transistors to a traditional SA, thus taking into consideration typical SA and cell areas [65], [66], our design increases the overall crossbar area by less than 1%. MATSA controllers are implemented as small finite-state machines whose area is negligible compared to the memory arrays.

C. SYSTEM EVALUATION

1) MATSA-EMBEDDED AND MATSA-PORTABLE

We compare the performance of MATSA-Embedded (32K compute-enabled columns) and MATSA-Portable (256K compute-enabled columns) with `cpuarm`, `cpui7`, and `fpga` baselines in Figure 14a. The smallest version, MATSA-Embedded, provides 30.20×/1.30×/8.14× lower execution times than `cpuarm`, `cpui7`, and `fpga`, respectively.

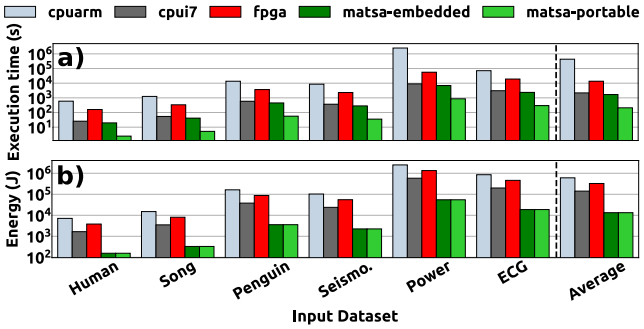

FIGURE 14. Latency and energy consumption of MATSA-Embedded (num_cols = 32K) and MATSA-Portable (num_cols = 256K) versus baselines (rd_lat = 5ns, wr_lat = 10ns, rd_en = 50nJ, wr_en = 70nJ).

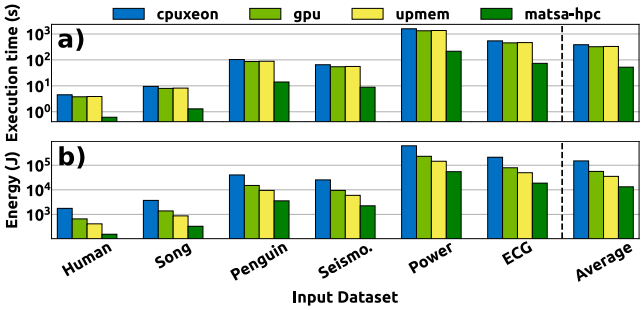

FIGURE 15. Execution times and energy consumption of MATSA-HPC (num_cols = 1M) versus baselines (rd_lat = 5ns, wr_lat = 10ns, rd_en = 50nJ, wr_en = 70nJ).

MATSA-Portable is further able to improve the performance by 241.66×/10.40×/65.28× with respect to the same baselines, respectively. These performance improvements stem from the higher available parallelism in PUM, where all compute-enable columns can compute independently. Next, we compare the energy consumption of MATSA-Embedded and MATSA-Portable with the same baselines in Figure 14b. MATSA-Embedded reduces the energy consumption by 45.67×/10.64×/24.58× with respect to `cpuarm`, `cpui7` and `fpga` baselines, respectively. We observe that 1) the energy reduction comes from eliminating the expensive off-chip data movement and 2) MATSA-Portable reduces the energy consumption by roughly the same factor as MATSA-Embedded. We deduce from these results that scaling MATSA improves the performance but does not penalize the energy efficiency.

2) MATSA-HPC

We first perform a performance comparison of MATSA-HPC and present the results in Figure 15a. We observe that MATSA-HPC achieves 7.3×/6.15×/6.3× lower execution times than `cpuxeon`, `gpu` and `upmem`, respectively, owing to enormous available parallelism (one million compute columns). Second, we compare the energy consumption of MATSA-HPC in Figure 15b and observe that it provides 11.29×/4.21×/2.65× lower energy consumption than `cpuxeon`, `gpu` and `upmem`, respectively. The energy efficiency benefits of MATSA-HPC

stem from the elimination of the off-chip data movements. We note that `cpuxeon` is bottlenecked by 1) the limited parallelism (number of cores) and 2) the high data movement costs through the memory hierarchy. The `gpu` baseline provides high parallelism but is limited by data movement from and to memory. The PNM-based `upmem` baseline provides high parallelism and lowers data access costs compared to CPU and GPUs. However, the sDTW kernel is compute-bound in `upmem` due to small general-purpose cores, in contrast to MATSA, a dedicated accelerator design for the sDTW kernel.

3) MATSA BENEFITS

Table 6 summarizes MATSA's benefits.

TABLE 6. MATSA's speedup and energy over baselines.

MATSA Version	Baseline	Speedup	Energy Savings
Embedded	cpuarm	30.20×	45.67×
Portable	cpui7	10.41×	10.65×
	FPGA	65.01×	24.58×
HPC	Xeon	7.35×	11.29×
	UPMEM	6.31×	2.65×
	GPU	6.15×	4.21×

V. RELATED WORK

To our knowledge, MATSA is the first sDTW accelerator via MRAM-based PUM. We compare extensively to CPU, GPU, FPGA, and state-of-the-art PNM platforms in section IV. In this section, we describe related works focusing on accelerating sDTW and prior PUM-based accelerators.

A. ACCELERATING DYNAMIC TIME WARPING (DTW)

Several works attempt to accelerate the sDTW kernel using GPUs [52], [67] and FPGAs [68]. section IV demonstrates that MATSA improves upon the performance of GPUs and FPGAs by 6.15× and 65.28× respectively, and supports arbitrary-sized datasets, a key drawback of prior work.

B. PROCESSING NEAR/USING MEMORY

There has been a significant interest in Processing-[Near/Using]-Memory-based solutions for overcoming the von Neumann bottleneck in modern computation platforms [5], [8], [15], [69], [70], [71], [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [82], [83], [84], [85], [86], [87], [88], [89], [90] for various applications using accelerators or general-purpose cores. In [91], ARM cores are used as NDP compute units to improve data analytics operators (e.g., group, join, sort). IMPICA [92] is an NDP pointer chasing accelerator. Tesseract [93] is a scalable NDP accelerator for parallel graph processing. TETRIS [94] is an NDP neural network accelerator. Lee et al. [95] propose an NDP accelerator for similarity search. GRIM-Filter [77] is an NDP accelerator for pre-alignment filtering in genome analysis. Boroumand et al. [9] analyze the energy and performance impact of data movement for several widely-used Google consumer workloads, providing NDP accelerators for them. CoNDA [70] provides efficient cache coherence support for NDP accelerators. SparseP [96] provides efficient data partitioning/maping techniques of the SpMV kernel tailored for near-bank NDP architectures. NDC is an NDP architecture [97] that has been proposed for MapReduce-style applications. Xu et al. [98] propose a memristor-based accelerator for accelerating the sDTW kernel. Despite promising performance, they do not discuss endurance challenges associated with memristors that restrict the lifetime of the accelerator. In contrast, MATSA considers this challenge and offers a usable lifetime of several decades. Chen and Gu [99] propose an sDTW accelerator that exploits DTW pipelining using a specially designed time flip-flop. Although this work uses memristors for computation, they do not leverage PUM. The data must be moved from/to memory (i.e., memristors do not store the data). In contrast, MATSA eliminates off-chip data movement to obtain high performance and energy efficiency.

VI. CONCLUSION

This paper presents MATSA, the first MRAM-based Accelerator for Time Series Analysis. The key idea is to exploit magnetoresistive crossbars to enable energy-efficient and fast time series computation in memory. MATSA provides the following key benefits: 1) significantly higher parallelism exploiting column-level bitwise operations, and 2) reduction in data movement overheads by leveraging PUM. MATSA improves performance and energy consumption over CPU, GPU, FPGA, and PNM platforms.

ACKNOWLEDGMENT

(Christina Giannoula and Aditya Manglik contributed equally to this work.)

REFERENCES

[1] P. Esling and C. Agon, "Time-series data mining," *ACM Comput. Surv. (CSUR)*, vol. 45, no. 1, pp. 1–34, 2012.

[2] A. Mueen and E. Keogh, "Extracting optimal performance from dynamic time warping," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 2129–2130.

[3] X. Yao and H.-L. Wei, "A modified dynamic time warping (MDTW) and innovative average non-self match distance (ANSD) method for anomaly detection in ECG recordings," in *Recent Advances in AI-enabled Automated Medical Diagnosis*. Boca Raton, FL, USA: CRC Press, 2022.

[4] S. Alaee, R. Mercer, K. Kamgar, and E. Keogh, "Time series motifs discovery under DTW allows more robust discovery of conserved structure," *Data Mining Knowl. Discovery*, vol. 35, no. 3, pp. 863–910, May 2021.

[5] J. Gómez-Luna, I. E. Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking a new paradigm: Experimental analysis and characterization of a real processing-in-memory system," *IEEE Access*, vol. 10, pp. 52565–52608, 2022.

[6] C. Giannoula, I. Fernandez, J. G. Luna, N. Koziris, G. Goumas, and O. Mutlu, "SparseP: Towards efficient sparse matrix vector multiplication on real processing-in-memory architectures," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 1, pp. 1–49, 2022.

[7] O. Mutlu, S. Ghose, J. Gomez-Luna, and R. Ausavarungnirun, "A modern primer on processing in memory," in *Emerging Computing: From Devices to Systems*. Berlin, Germany: Springer, 2022, pp. 171–243.

[8] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Processing data where it makes sense: Enabling in-memory computation," *Microprocess. Microsyst.*, vol. 67, pp. 28–41, Jun. 2019.

[9] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proc. ASPLOS*, 2018, pp. 316–331.

[10] F. Devaux, "The true processing in memory accelerator," in *Proc. IEEE Hot Chips 31 Symp. (HCS)*, Aug. 2019, pp. 1–24.

[11] S. Angizi, J. Sun, W. Zhang, and D. Fan, "AlignS: A processing-in-memory accelerator for DNA short read alignment leveraging SOT-MRAM," in *Proc. DAC*, 2019, pp. 1–6.

[12] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: A DRAM-based reconfigurable in-situ accelerator," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2017, pp. 288–301.

[13] S. Angizi, Z. He, A. Awad, and D. Fan, "MRIMA: An MRAM-based in-memory accelerator," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 5, pp. 1123–1136, May 2020.

[14] S. Angizi, Z. He, A. S. Rakin, and D. Fan, "CMP-PIM: An energy-efficient comparator-based processing-in-memory neural network accelerator," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[15] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2017, pp. 273–287.

[16] K. Roy, I. Chakraborty, M. Ali, A. Ankit, and A. Agrawal, "In-memory computing in emerging memory technologies for machine learning: An overview," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[17] H. Lin, X. Luo, L. Liu, D. Wang, X. Zhao, Z. Wang, X. Xue, F. Zhang, and G. Xing, "All-electrical control of compact SOT-MRAM: Toward highly efficient and reliable non-volatile in-memory computing," *Micromachines*, vol. 13, no. 2, p. 319, Feb. 2022.

[18] S. Yu and P.-Y. Chen, "Emerging memory technologies: Recent trends and prospects," *IEEE Solid StateCircuits Mag.*, vol. 8, no. 2, pp. 43–56, Spring 2016.

[19] W. J. Gallagher, E. Chien, T.-W. Chiang, J.-C. Huang, M.-C. Shih, C. Y. Wang, C. Bair, G. Lee, Y.-C. Shih, C.-F. Lee, R. Wang, K.-H. Shen, J. J. Wu, W. Wang, and H. Chuang, "Recent progress and next directions for embedded MRAM technology," in *Proc. Symp. VLSI Technol.*, Jun. 2019, pp. 190–191.

[20] P. Patel, E. Keogh, J. Lin, and S. Lonardi, "Mining motifs in massive time series databases," in *Proc. ICDM*, 2002, pp. 370–377.

[21] E. Keogh, J. Lin, S.-H. Lee, and H. V. Herle, "Finding the most unusual time series subsequence: Algorithms and applications," *Knowl. Inf. Syst.*, vol. 11, no. 1, pp. 1–27, Dec. 2006.

[22] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh, "Matrix profile I: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets," in *Proc. ICDM*, 2016, pp. 1317–1322.

[23] C. A. Ratanamahatana and E. Keogh, "Making time-series classification more accurate using learned constraints," in *Proc. SIAM Int. Conf. Data Mining*, Apr. 2004, pp. 11–22.

[24] R. H. Shumway and D. S. Stoffer, *Time Series Analysis and Its Applications: With R Examples*. New York, NY, USA: Springer, 2017.

[25] A. E. X. Brown, E. I. Yemini, L. J. Grundy, T. Jucikas, and W. R. Schafer, "A dictionary of behavioral motifs reveals clusters of genes affecting caenorhabditis elegans locomotion," *Proc. Nat. Acad. Sci. USA*, vol. 110, no. 2, pp. 791–796, Jan. 2013.

[26] A. Balasubramanian, J. Wang, and B. Prabhakaran, "Discovering multidimensional motifs in physiological signals for personalized healthcare," *IEEE J. Sel. Topics Signal Process.*, vol. 10, no. 5, pp. 832–841, Aug. 2016.

[27] Y. Tanaka, K. Iwamoto, and K. Uehara, "Discovery of time-series motif from multi-dimensional data based on MDL principle," *Mach. Learn.*, vol. 58, nos. 2–3, pp. 269–300, Feb. 2005.

[28] A. McGovern, D. H. Rosendahl, R. A. Brown, and K. K. Droegemeier, "Identifying predictive multi-dimensional time series motifs: An application to severe weather prediction," *Data Mining Knowl. Discovery*, vol. 22, nos. 1–2, pp. 232–258, Jan. 2011.

[29] B. Szigeti, A. Deogade, and B. Webb, "Searching for motifs in the behaviour of larval Drosophila melanogaster and caenorhabditis elegans reveals continuity between behavioural states," *J. Roy. Soc. Interface*, vol. 12, no. 113, Dec. 2015, Art. no. 20150899.

[30] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering—A decade review," *Inf. Syst.*, vol. 53, pp. 16–38, Oct. 2015.

[31] C. McKee, I. Itikarai, and H. Davies, "Instrumental volcano surveillance and community awareness in the lead-up to the 1994 eruptions at Rabaul, Papua New Guinea," in *Observing the Volcano World*. Berlin, Germany: Springer, 2018, pp. 205–233.

[32] E. Philip Howrey, "The role of time series analysis in econometric model evaluation," in *Evaluation of Econometric Models*. Cambridge, MA, USA: Academic, 1980, pp. 275–307.

[33] R. H. Shumway, *Applied Statistical Time Series Analysis*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1988.

[34] K. Nakagawa, M. Imamura, and K. Yoshida, "Stock price prediction using k-medoids clustering with indexing dynamic time warping," *Electron. Commun. Jpn.*, vol. 102, no. 2, pp. 3–8, Feb. 2019.

[35] D. Barber, A. T. Cemgil, and S. Chiappa, *Bayesian Time Series Models*. Cambridge, U.K.: Cambridge Univ. Press, 2011.

[36] S. A. P. Kumar and P. K. Bora, "Time series analysis and signal processing," in *Proc. 2nd Nat. Conf. Comput. Intell. Signal Process. (CISP)*, Mar. 2012, p. 24.

[37] B. Wu, "Pattern recognition and classification in time series analysis," *Appl. Math. Comput.*, vol. 62, no. 1, pp. 29–45, Apr. 1994.

[38] A. Lakhina, M. Crovella, and C. Diot, "Characterization of network-wide anomalies in traffic flows," in *Proc. 4th ACM SIGCOMM Conf. Internet Meas.*, Oct. 2004, pp. 201–206.

[39] T. Dunn, H. Sadasivan, J. Wadden, K. Goliya, K.-Y. Chen, D. Blaauw, R. Das, and S. Narayanasamy, "SquiggleFilter: An accelerator for portable virus detection," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2021, pp. 535–549.

[40] R. Vio, N. R. Kristensen, H. Madsen, and W. Wamsteker, "Time series analysis in astronomy: Limits and potentialities," *Astron. Astrophys.*, vol. 435, no. 2, pp. 773–780, May 2005.

[41] K. Nusratullah, S. A. Khan, A. Shah, and W. H. Butt, "Detecting changes in context using time series analysis of social network," in *Proc. SAI Intell. Syst. Conf. (IntelliSys)*, Nov. 2015, pp. 996–1001.

[42] E. Keogh and S. Kasetty, "On the need for time series data mining benchmarks: A survey and empirical demonstration," *Data Mining Knowl. Discovery*, vol. 7, no. 4, pp. 349–371, Oct. 2003.

[43] A. Christophersen, N. I. Deligne, A. M. Hanea, L. Chardot, N. Fournier, and W. P. Aspinall, "Bayesian network modeling and expert elicitation for probabilistic eruption forecasting: Pilot study for Whakaari/White Island, New Zealand," *Frontiers Earth Sci.*, vol. 6, p. 211, Nov. 2018.

[44] A. Klos, M. S. Bos, and J. Bogusz, "Detecting time-varying seasonal signal in GPS position time series with different noise levels," *GPS Solutions*, vol. 22, no. 1, pp. 1–11, Jan. 2018.

[45] R. Stoermer, R. Mager, A. Roessler, F. Mueller-Spahn, and A. H. Bullinger, "Monitoring human-virtual reality interaction: A time series analysis approach," *CyberPsychology Behav.*, vol. 3, no. 3, pp. 401–406, Jun. 2000.

[46] A. D. Calin, "Gesture recognition on Kinect time series data using dynamic time warping and hidden Markov models," in *Proc. 18th Int. Symp. Symbolic Numeric Algorithms Scientific Comput. (SYNASC)*, Sep. 2016, pp. 264–271.

[47] S. Ayhan and H. Samet, "Time series clustering of weather observations in predicting climb phase of aircraft trajectories," in *Proc. 9th ACM SIGSPATIAL Int. Workshop Comput. Transp. Sci.*, Oct. 2016, pp. 25–30.

[48] L. Li, X. Su, Y. Zhang, Y. Lin, and Z. Li, "Trend modeling for traffic time series analysis: An integrated study," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 6, pp. 3430–3439, Dec. 2015.

[49] S. L. Hyland, M. Faltys, M. Hüser, X. Lyu, T. Gumbsch, C. Esteban, C. Bock, M. Horn, M. Moor, B. Rieck, M. Zimmermann, D. Bodenham, K. Borgwardt, G. Rätsch, and T. M. Merz, "Early prediction of circulatory failure in the intensive care unit using machine learning," *Nature Med.*, vol. 26, no. 3, pp. 364–373, Mar. 2020.

[50] G. Chen, G. Lu, Z. Xie, and W. Shang, "Anomaly detection in EEG signals: A case study on similarity measure," *Comput. Intell. Neurosci.*, vol. 2020, pp. 1–16, Jan. 2020.

[51] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," in *Proc. KDD Workshop*, 1994, vol. 10, no. 16, pp. 359–370.

[52] B. Schmidt and C. Hundt, "cuDTW++: Ultra-fast dynamic time warping on CUDA-enabled GPUs," in *Euro-Par 2020: Parallel Processing*. Berlin, Germany: Springer, 2020, pp. 597–612.

[53] *NVIDIA Visual Profiler*. Accessed: Nov. 16, 2023. [Online]. Available: https://developer.nvidia.com/nvidia-visual-profiler
[54] *Introduction to UPMEM PIM. Processing-in-Memory (PIM) on DRAM Accelerator*, UPMEM, Grenoble, France, 2018.
[55] N. Hajinazar, G. F. Oliveira, S. Gregorio, J. D. Ferreira, N. M. Ghiasi, M. Patel, M. Alser, S. Ghose, J. Gómez-Luna, and O. Mutlu, "SIMDRAM: A framework for bit-serial SIMD processing using DRAM," in *Proc. 26th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Apr. 2021, pp. 329–345.
[56] P. Zuo, Y. Hua, M. Zhao, W. Zhou, and Y. Guo, "Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2018, pp. 442–454.
[57] T. Daulby, A. Savanth, A. S. Weddell, and G. V. Merrett, "Comparing NVM technologies through the lens of intermittent computation," in *Proc. 8th Int. Workshop Energy Harvesting Energy-Neutral Sens. Syst.*, Nov. 2020, pp. 77–78.
[58] S. Mittal, "A survey of ReRAM-based architectures for processing-in-memory and neural networks," *Mach. Learn. Knowl. Extraction*, vol. 1, no. 1, pp. 75–114, Apr. 2018.
[59] Y. Zhang, J. Wang, C. Lian, Y. Bai, G. Wang, Z. Zhang, Z. Zheng, L. Chen, K. Zhang, G. Sirakoulis, and Y. Zhang, "Time-domain computing in memory using spintronics for energy-efficient convolutional neural network," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 3, pp. 1193–1205, Mar. 2021.
[60] X. Jin, W. Chen, X. Li, N. Yin, C. Wan, M. Zhao, X. Han, and Z. Yu, "High-reliability, reconfigurable, and fully non-volatile full-adder based on SOT-MTJ for image processing applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 70, no. 2, pp. 781–785, Feb. 2023.
[61] J. Wang, Y. Bai, H. Wang, Z. Hao, G. Wang, K. Zhang, Y. Zhang, W. Lv, and Y. Zhang, "Reconfigurable bit-serial operation using toggle SOT-MRAM for high-performance computing in memory architecture," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 11, pp. 4535–4545, Nov. 2022.
[62] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MAGIC—Memristor-aided logic," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 11, pp. 895–899, Nov. 2014.
[63] (2022). *ZSim+Ramulator*. [Online]. Available: github.com/CMU-SAFARI/ramulator-pim
[64] R. Saha, Y. P. Pundir, and P. Kumar Pal, "Comparative analysis of STT and SOT based MRAMs for last level caches," *J. Magn. Magn. Mater.*, vol. 551, Jun. 2022, Art. no. 169161.
[65] M. Uddin and G. S. Rose, "A practical sense amplifier design for memristive crossbar circuits (PUF)," in *Proc. 31st IEEE Int. Syst.-Chip Conf. (SOCC)*, Sep. 2018, pp. 209–214.
[66] Y. Seo, K.-W. Kwon, and K. Roy, "Area-efficient SOT-MRAM with a Schottky diode," *IEEE Electron Device Lett.*, vol. 37, no. 8, pp. 982–985, Aug. 2016.
[67] H. Sadasivan, D. Stiffler, A. Tirumala, J. Israeli, and S. Narayanasamy, "Accelerated dynamic time warping on GPU for selective nanopore sequencing," *BioRxiv*, Mar. 2023.
[68] Z. Wang, S. Huang, L. Wang, H. Li, Y. Wang, and H. Yang, "Accelerating subsequence similarity search based on dynamic time warping distance with FPGA," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Array*, 2013, pp. 53–62.
[69] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu, "Processing-in-memory: A workload-driven perspective," *IBM J. Res. Develop.*, vol. 63, no. 6, pp. 1–19, Nov. 2019.
[70] A. Boroumand, S. Ghose, M. Patel, H. Hassan, B. Lucia, R. Ausavarungnirun, K. Hsieh, N. Hajinazar, K. T. Malladi, H. Zheng, and O. Mutlu, "CoNDA: Efficient cache coherence support for near-data accelerators," in *Proc. ACM/IEEE 46th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2019, pp. 629–642.
[71] G. Singh, D. Diamantopoulos, C. Hagleitner, J. Gomez-Luna, S. Stuijk, O. Mutlu, and H. Corporaal, "NERO: A near high-bandwidth memory stencil accelerator for weather prediction modeling," in *Proc. 30th Int. Conf. Field-Programmable Log. Appl. (FPL)*, Aug. 2020, pp. 9–17.
[72] D. Fujiki, S. Mahlke, and R. Das, "Duality cache for data parallel acceleration," in *Proc. ACM/IEEE 46th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2019, pp. 1–14.
[73] C. Giannoula, N. Vijaykumar, N. Papadopoulou, V. Karakostas, I. Fernandez, J. Gómez-Luna, L. Orosa, N. Koziris, G. Goumas, and O. Mutlu, "SynCron: Efficient synchronization support for near-data-processing architectures," in *Proc. IEEE Int. Symp. High-Performance Comput. Archit. (HPCA)*, Feb. 2021, pp. 263–276.
[74] H. S. Stone, "A logic-in-memory computer," *IEEE Trans. Comput.*, vol. C-19, no. 1, pp. 73–78, Jan. 1970.
[75] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 383–396.
[76] I. Fernandez, R. Quislant, E. Gutiérrez, O. Plata, C. Giannoula, M. Alser, J. Gómez-Luna, and O. Mutlu, "NATSA: A near-data processing accelerator for time series analysis," in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, Oct. 2020, pp. 120–129.
[77] J. S. Kim, D. Senol Cali, H. Xin, D. Lee, S. Ghose, M. Alser, H. Hassan, O. Ergin, C. Alkan, and O. Mutlu, "GRIM-filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies," *BMC Genomics*, vol. 19, no. 2, pp. 23–40, May 2018.
[78] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2015, pp. 336–348.
[79] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, "Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–13.
[80] B. Gu, A. S. Yoon, D.-H. Bae, I. Jo, J. Lee, J. Yoon, J.-U. Kang, M. Kwon, C. Yoon, S. Cho, J. Jeong, and D. Chang, "Biscuit: A framework for near-data processing of big data workloads," in *Proc. ISCA*, 2016, pp. 153–165.
[81] M. Hashemi, O. Mutlu, and Y. N. Patt, "Continuous runahead: Transparent hardware acceleration for memory intensive workloads," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.
[82] K. Hsieh, E. Ebrahim, G. Kim, N. Chatterjee, M. O'Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, "Transparent offloading and mapping (TOM): Enabling programmer-transparent near-data processing in GPU systems," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 204–216.
[83] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory," in *Proc. ISCA*, 2016, pp. 380–392.
[84] G. Kim, N. Chatterjee, M. O'Connor, and K. Hsieh, "Toward standardized near-data processing with unrestricted data placement for GPUs," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2017, pp. 1–12.
[85] J. H. Lee, J. Sim, and H. Kim, "BSSync: Processing near memory for machine learning workloads with bounded staleness consistency models," in *Proc. Int. Conf. Parallel Archit. Compilation (PACT)*, Oct. 2015, pp. 241–252.
[86] Z. Liu, I. Calciu, M. Herlihy, and O. Mutlu, "Concurrent data structures for near-memory computing," in *Proc. 29th ACM Symp. Parallelism Algorithms Architectures*, Jul. 2017, pp. 235–245.
[87] O. Mutlu and L. Subramanian, "Research problems and opportunities in memory systems," *Supercomputing Frontiers Innov.*, vol. 1, no. 3, pp. 19–55, 2014.
[88] S. Jain, S. Sapatnekar, J.-P. Wang, K. Roy, and A. Raghunathan, "Computing-in-memory with spintronics," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 1640–1645.
[89] N. M. Ghiasi, J. Park, H. Mustafa, J. Kim, A. Olgun, A. Gollwitzer, D. S. Cali, C. Firtina, H. Mao, and N. A. Alserr, "GenStore: A high-performance and energy-efficient in-storage computing system for genome sequence analysis," in *Proc. ASPLOS*, 2022, pp. 1–20.
[90] G. F. Oliveira, J. Gómez-Luna, L. Orosa, S. Ghose, N. Vijaykumar, I. Fernandez, M. Sadrosadati, and O. Mutlu, "DAMOV: A new methodology and benchmark suite for evaluating data movement bottlenecks," *IEEE Access*, vol. 9, pp. 134457–134502, 2021.
[91] M. Drumond, A. Daglis, N. Mirzadeh, D. Ustiugov, J. Picorel, B. Falsafi, B. Grot, and D. Pnevmatikatos, "The Mondrian data engine," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 639–651.

[92] K. Hsieh, S. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, "Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation," in *Proc. IEEE 34th Int. Conf. Comput. Design (ICCD)*, Oct. 2016, pp. 25–32.
[93] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2015, pp. 105–117.
[94] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and efficient neural network acceleration with 3D memory," in *Proc. ASPLOS*, 2017, pp. 751–764.
[95] V. T. Lee, A. Mazumdar, C. C. del Mundo, A. Alaghi, L. Ceze, and M. Oskin, "Application codesign of near-data processing for similarity search," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2018, pp. 896–907.
[96] C. Giannoula, I. Fernandez, J. Gomez-Luna, N. Koziris, G. Goumas, and O. Mutlu, "SparseP: Efficient sparse matrix vector multiplication on real processing-in-memory architectures," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2022, pp. 288–291.
[97] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, "NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Mar. 2014, pp. 190–200.
[98] X. Xu, F. Lin, A. Wang, X. Yao, Q. Lu, W. Xu, Y. Shi, and Y. Hu, "Accelerating dynamic time warping with memristor-based customized fabrics," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 4, pp. 729–741, Apr. 2018.
[99] Z. Chen and J. Gu, "High-throughput dynamic time warping accelerator for time-series classification with pipelined mixed-signal time-domain computing," *IEEE J. Solid-State Circuits*, vol. 56, no. 2, pp. 624–635, Feb. 2021.

IVAN FERNANDEZ received the B.S. degree in computer engineering, the M.S. degree in mechatronics engineering, and the Ph.D. degree from the University of Málaga, in 2017, 2018, and 2023, respectively. During the Ph.D. degree, he stayed for ten months with SAFARI Research Group led by Onur Mutlu (ETH Zürich) as an affiliated Researcher. In 2022, he joined Universitat Politècnica de Catalunya and Barcelona Supercomputing Center as a Visiting Researcher, where he is currently involved in several international projects, such as European Processor Initiative. His current research interests include processing in memory, near-data processing, stacked memory architectures, high-performance computing and transprecision computing applied to time series analysis, and bioinformatics.

CHRISTINA GIANNOULA received the Ph.D. degree from the School of Electrical and Computer Engineering, National Technical University of Athens, advised by Prof. Georgios Goumas, Prof. Nectarios Koziris, and Prof. Onur Mutlu, in October 2022. She is currently a Postdoctoral Researcher with the University of Toronto working with Prof. Gennady Pekhimenko and his research group. She is also with the SAFARI Research Group and Prof. Onur Mutlu. She has several publications and awards for her research on the aforementioned topics. Her research interests include the intersection of computer architecture, computer systems, and high-performance computing. Specifically, her research focuses on the hardware/software co-design of emerging applications, including graph processing, pointer-chasing data structures, machine learning workloads, and sparse linear algebra, with modern computing paradigms, such as large-scale multicore systems, disaggregated memory systems, and near-data processing architectures. She is a member of ACM, ACM-W, and the Technical Chamber of Greece. For more information, please see her webpage at https://cgiannoula.github.io/.

ADITYA MANGLIK received the bachelor's degree in electrical engineering from BITS Pilani, in 2018, and the master's degree in electrical engineering from ETH Zürich, Switzerland, in 2024. He is currently a Graduate Student with the SAFARI Research Group, ETH Zürich, advised by Prof. Onur Mutlu. His research interests include sustainable software, energy-efficient systems, secure system design, and processing-in-memory. For more information, please see his webpage at https://adityamanglik.github.io/.

RICARDO QUISLANT received the M.Sc. degree in computer engineering from the University of Granada, in 2006, and the Ph.D. degree from the University of Málaga, Spain, in 2012. Currently, he is an Assistant Professor with the Department of Computer Architecture, University of Málaga. His main research interests include computer memory systems and high-performance computing, with special regard to transactional memory.

NIKA MANSOURI GHIASI received the B.S. degree in electrical engineering from the University of Tehran and the M.S. degree in electrical engineering from ETH Zürich, where she is currently pursuing the Ph.D. degree. She is with ETH Zürich advised by Onur Mutlu. Her research interests include emerging memory and processing technologies, near-data processing, storage systems, and bioinformatics.

JUAN GÓMEZ-LUNA (Member, IEEE) received the B.S. and M.S. degrees in telecommunication engineering from the University of Seville, Spain, in 2001, and the Ph.D. degree in computer science from the University of Córdoba, Spain, in 2012. From 2005 to 2017, he was a Faculty Member with the University of Córdoba. He is currently a Senior Researcher and a Lecturer with the SAFARI Research Group, ETH Zürich. His research interests include processing-in-memory, memory systems, heterogeneous computing and hardware and software acceleration of medical imaging, and bioinformatics. He is the Lead Author of PrIM (https://github.com/CMU-SAFARI/prim-benchmarks), the first publicly-available benchmark suite for a real-world processing-in-memory architecture; and Chai (https://github.com/chai-benchmarks/chai), a benchmark suite for heterogeneous systems with CPU/GPU/FPGA.

ELADIO GUTIERREZ received the M.Sc. and Ph.D. degrees in telecommunication engineering from the University of Málaga, Spain, in 1995 and 2001, respectively. Since 2003, he has been an Associate Professor with the Department of Computer Architecture, University of Málaga. His research interests include parallel architectures and algorithms, graphics processing units, and automatic parallelization.

OSCAR PLATA received the M.S. and Ph.D. degrees in physics from University of Santiago de Compostela, Spain, in 1985 and 1989, respectively. He started as an Assistant Professor with University of Santiago de Compostela, where he became an Associated Professor, in 1990. He moved to the University of Málaga, in 1995, where he became a Full Professor with the Computer Architecture Department, in 2002. His research interests include related to high-performance computing and parallel architectures.

ONUR MUTLU (Fellow, IEEE) received the B.S. degree in computer engineering and psychology from the University of Michigan, Ann Arbor, MI, USA, and the M.S. and Ph.D. degrees in ECE from The University of Texas at Austin. He started the Computer Architecture Group, Microsoft Research, from 2006 to 2009, and held various product and research positions with Intel Corporation, Advanced Micro Devices, VMware, and Google. He is currently a Professor of computer science with ETH Zürich. He is also a Faculty Member with Carnegie Mellon University, where he previously held the Strecker Early Career Professorship. His current broader research interests include computer architecture, systems, hardware security, and bioinformatics. A variety of techniques, he along with his group and collaborators, has invented over the years have influenced industry and have been employed in commercial microprocessors and memory/storage systems. He received the IEEE High Performance Computer Architecture Test of Time Award, the IEEE Computer Society Edward J. McCluskey Technical Achievement Award, the ACM SIGARCH Maurice Wilkes Award, the Inaugural IEEE Computer Society Young Computer Architect Award, the Inaugural Intel Early Career Faculty Award, U.S. National Science Foundation CAREER Award, the Carnegie Mellon University Ladd Research Award, the faculty partnership awards from various companies, and a healthy number of best paper or ''Top Pick'' paper recognitions at various computer systems, architecture, and security venues. He is an ACM Fellow and an Elected Member of the Academy of Europe (Academia Europaea). His computer architecture and digital logic design course lectures and materials are freely available on YouTube (https://www.youtube.com/OnurMutluLectures). His research group makes a wide variety of software and hardware artifacts freely available online (https://safari.ethz.ch/). For more information, please see his webpage at https://people.inf.ethz.ch/omutlu/.