Accelerating Time Series Analysis via Processing using Non-Volatile Memories

Ivan Fernandez[§†¶] *Christina Giannoula[†‡] *Aditya Manglik[†] Ricardo Quislant[§] Nika Mansouri Ghiasi[†]
Juan Gómez-Luna[†] Eladio Gutierrez[§] Oscar Plata[§] Onur Mutlu[†]
[§]*University of Malaga* [†]*ETH Zürich* [¶]*Barcelona Supercomputing Center* [‡]*National Technical University of Athens*

***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.**

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 and GPU architectures, we comprehensively characterize the kernel's performance on these platforms (§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]–[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 (§II-D).

In contrast to PNM, *Processing-Using-Memory* (PUM) [7], [11]–[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 [16], 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 [17]. However, different NVM substrates exhibit varying latency, energy, and endurance characteristics, a key design constraint for different accelerators [18]. Magnetoresistive RAM (MRAM)-based PUM substrates offer low read/write latencies, low energy per operation, and high endurance [19], [20]. Considering these characteristics, 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.

*** Christina Giannoula and Aditya Manglik have equal contribution.**

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 [21], [22]. 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 & 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* [23] (similarities) and high distance as *discords* [24] (anomalies). The state-of-the-art set of tools to perform time series analysis is Matrix Profile [25] (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 [26]. 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.

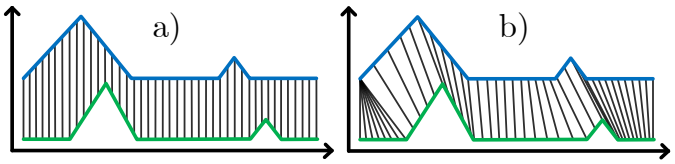

Fig. 1: Example of similarity calculation between two subsequences (blue and green). The one-to-one approach in a) provides a low similarity as it only compares each i^{th} point of blue with each i^{th} point of green. In contrast, DTW in b) successfully matches the points of the subsequences.

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 [27]: epidemiology, genomics, neuroscience, medicine, environmental sciences, economics, and many more. Table I presents a few examples for applications of TSA.

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

Field	References	Field	References
Bioinformatics	[28]–[30]	Speech Recognition	[31]
Robotics	[32], [33]	Weather Prediction	[34]
Neuroscience	[28], [35]	Entomology	[36]
Machine Learning	[37]–[39]	Geophysics	[40]–[43]
Econometrics	[44]	Statistics	[45]
Finance	[46], [47]	Control Engineering	[48]–[50]
Signal Processing	[51]	Pattern Recognition	[52]
Communication	[53]–[55]	Medicine	[56]–[59]
Astronomy	[60], [61]	Social Networks	[62], [63]
Clustering	[37], [38]	Classification	[39]
Earthquakes	[40]–[43]	GPS Tracking	[64]
Virtual Reality	[65]	Gesture Recognition	[66], [67]
Trajectories	[68]	Traffic Monitoring	[69]

TABLE I: Time Series Analysis main applications

be applied to earthquake relocation and volcano monitoring to help improve earthquake and volcanic hazard assessments.

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 [70]. 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 [59].
- **Earthquake Detection**. TSA is critical to process seismograph data and detect anomalies for further analysis [42].

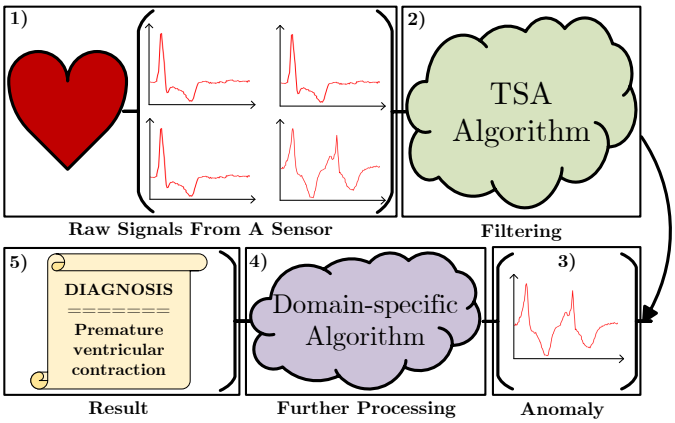

Fig. 2: Example TSA application, where TSA acts as a filter to avoid most of the computation. TSA selects the relevant queries (anomalies) and discards the irrelevant ones.

C. Dynamic Time Warping (DTW)

DTW algorithm was first introduced by [71]. 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:

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

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 [71]. Second, the distance value is added to the minimum of the three neighboring cells as follows:

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

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 [72]. 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.

CPU. We profile the performance of sDTW on an Intel Xeon Phi 7210 CPU using the Intel Advisor tool [73]. 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.

GPU. Several prior works propose accelerating sDTW using GPUs (e.g., [74]). 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 [75] 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.

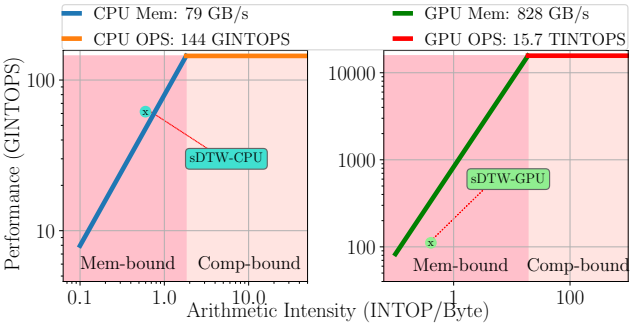

Fig. 3: Roofline plots for sDTW on a many-core CPU platform (left) and a server-class GPU (right).

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

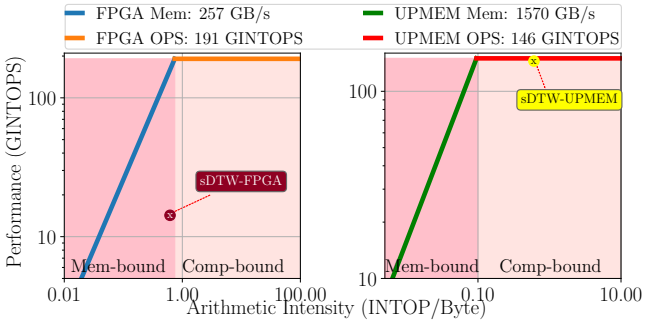

Fig. 4: Roofline plots for sDTW on FPGA (left) and UPMEM (right) platforms.

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.

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 [76], 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 (§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

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.*

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 [16], [77]. However, it is challenging to support frequent write operations as NVM-based PUM architectures due to significant write latency and low endurance [78]. Table II 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 (§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 [79]. 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.

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**	$> 10^{15}$

TABLE II: Characteristics of different NVM technologies [19].

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 [80] by exploiting matrix-vector multiplication. Other approaches can exploit bitwise operations with high performance and energy savings [81]–[83]. Figure 5-a shows a typical crossbar organization with memory cells connected using bitlines and wordlines.

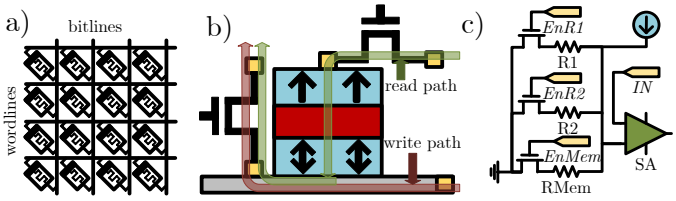

Fig. 5: a) Crossbar organization. b) Magneto-resistive cell. c) Reconfigurable SA that performs in-memory operations based on the voltage variations across the bitline.

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) [19], 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 [84] 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.

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.

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

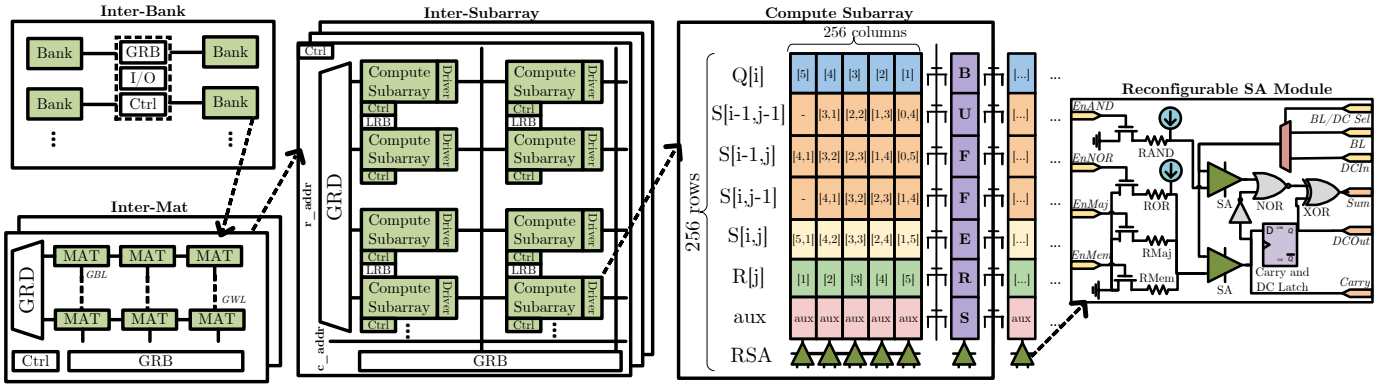

Fig. 6: MATSA's high-level architecture and data mapping flow.

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 §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 [85], 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).

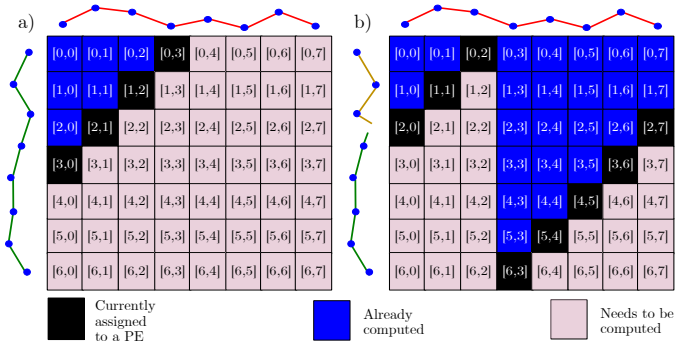

Fig. 7: Wavefront-based sDTW computation in MATSA.

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

F. Programming Interface & System Integration

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

```
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.

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.

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 x86 CPU @ 3.2GHz, 64KB L1, 256KB L2, 12MB L3 and 64GB DDR4.
- **CPU-Xeon (`cpuxeon`):** Two 18-core (36 threads) Intel Xeon Gold 6154 x86 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).
- **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).

Baselines. We use ZSim+Ramulator [86] 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.

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 [87] listed in Table III. Based on that, we conservatively select an operating point (highlighted in bold) for the evaluations taking into account realistic MRAM device progress projections [88]. 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.

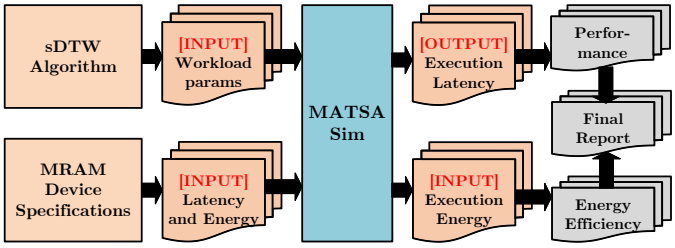

Fig. 8: Overview of MATSA Simulator.

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

TABLE III: MATSA design space exploration parameters.

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

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

TABLE IV: Workloads used in MATSA characterization.

.

Time Series	Reference Size	Query Size	Num. Queries
Human [89]	7997	120	128K
Song [90]	20234	200	64K
Penguin [91]	109842	800	32K
Seismology [90]	1727990	64	16K
Power [92]	1754985	1536	16K
ECG [93]	1800000	512	16K

TABLE V: Real-world workloads used in our evaluation.

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).

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.

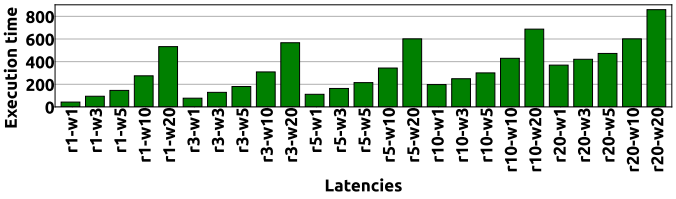

Fig. 9: Execution time when varying cell read and write latencies (ref_size=128K, query_size=8K, num_queries=8K, matsa_cols=128K).

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

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.

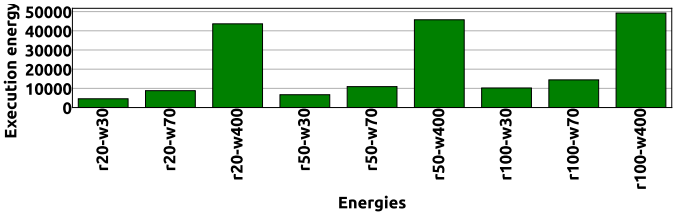

Fig. 10: Execution energy when varying cell read and write energies (ref_size=128K, query_size=8K, num_queries=8K, matsa_cols=128K).

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

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.

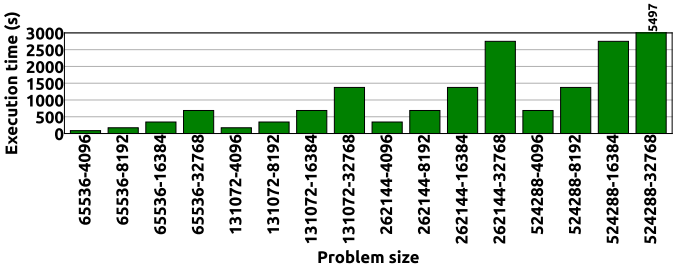

Fig. 11: Execution time when varying dataset sizes (num_queries=8K, matsa_cols=128K).

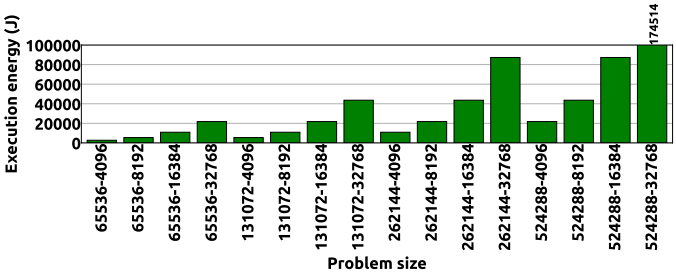

Fig. 12: Execution energy when varying dataset sizes (num_queries=8K, matsa_cols=128K).

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

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.

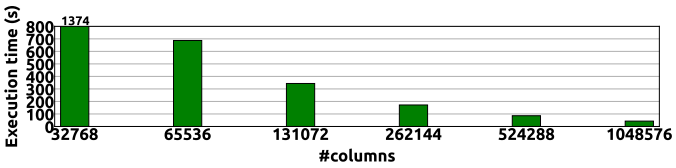

Fig. 13: Execution time when varying MATSA sizes.

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

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 $\approx 4\times10^9$ times. Based on Table II, 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.

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 [94], [95], 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

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. 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.

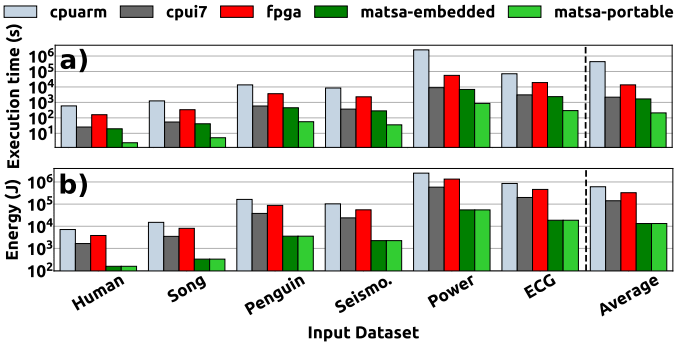

Fig. 14: Latency and energy consumption of MATSA-Embedded (num_cols=32K) and MATSA-Portable (num_cols=256K) versus baselines (rd_lat=5ns, wr_lat=10ns, rd_en=50nJ, wr_en=70nJ).

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,

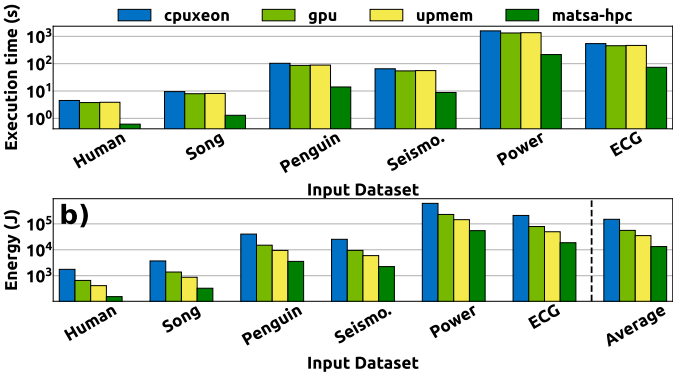

Fig. 15: Execution times and energy consumption of MATSA-HPC (num_cols=1M) versus baselines (rd_lat=5ns, wr_lat=10ns, rd_en=50nJ, wr_en=70nJ).

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.

MATSA Benefits. Table VI summarizes MATSA's benefits.

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×

TABLE VI: MATSA's Speedup and Energy over baselines.

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 §IV. In this section, we describe related works focusing on accelerating sDTW and prior PUM-based accelerators.

Accelerating Dynamic Time Warping (DTW). Several works attempt to accelerate the sDTW kernel using GPUs [74], [96] and FPGAs [97], [98]. §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.

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 [6], [8], [12], [14]–[16], [76], [84], [99]–[229] for various applications using accelerators or general-purpose cores. In [174], ARM cores are used as NDP compute units to improve data analytics operators (e.g., group, join, sort). IMPICA [230] is an NDP pointer chasing accelerator. Tesseract [231] is a scalable NDP accelerator for parallel graph processing. TETRIS [173] is an NDP neural network accelerator. Lee et al. [232] propose an NDP accelerator for similarity search. GRIM-Filter [146] is an NDP accelerator for pre-alignment filtering [233]–[237] in genome analysis [238]. 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 [150] provides efficient cache coherence support for NDP accelerators. SparseP [239], [240] provides efficient data partitioning/maping techniques of the SpMV kernel tailored for near-bank NDP architectures. Finally, an NDP architecture [169] has been proposed for MapReduce-style applications.. Xu et al. [241] 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 [242] 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. Conclusions

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.

Acknowledgements

This work has been supported by TIN2016-80920-R and UMA18-FEDERJA-197 Spanish projects, and HiPEAC collaboration grants. We also acknowledge support from the SAFARI Group's industrial partners, especially ASML, Facebook, Google, Huawei, Intel, Microsoft, and VMware, as well as support from Semiconductor Research Corporation.

References

[1] Philippe Esling and Carlos Agon. Time-series Data Mining. *ACM CSUR*, 2012.

[2] Abdullah Mueen and Eamonn Keogh. Extracting Optimal Performance from Dynamic Time Warping. In *SIGKDD*, 2016.

[3] Xinxin Yao and Hua-Liang 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*. CRC Press, 2022.

[4] Sara Alaee, Ryan Mercer, Kaveh Kamgar, and Eamonn Keogh. Time Series Motifs Discovery Under DTW Allows More Robust Discovery of Conserved Structure. *Data Mining and Knowledge Discovery*, 35(3):863–910, 2021.

[5] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System. *IEEE Access*, 10:52565–52608, 2022.

[6] Christina Giannoula and Ivan Fernandez and Juan Gómez-Luna and Nectarios Koziris and Georgios Goumas and Onur Mutlu. SparseP: Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-In-Memory Architectures. *Proc. ACM Meas. Anal. Comput. Syst.*, 6(1), 2022.

[7] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. A Modern Primer On Processing In Memory. In *Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann*, pages 171–243. Springer, 2022.

[8] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. Processing Data Where it Makes Sense: Enabling In-Memory Computation. *Microprocessors and Microsystems*, 2019.

[9] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu. Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks. *ASPLOS*, 2018.

[10] Fabrice Devaux. The True Processing in Memory Accelerator. In *2019 IEEE Hot Chips 31 Symposium (HCS)*, pages 1–24. IEEE Computer Society, 2019.

[11] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. AlignS: A Processing-in-Memory Accelerator for DNA Short Read Alignment Leveraging SOT-MRAM. In *DAC*, pages 1–6. IEEE, 2019.

[12] Shuangchen Li, Dimin Niu, Krishna T Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. DRISA: A DRAM-based Reconfigurable In-situ Accelerator. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 288–301. IEEE, 2017.

[13] Shaahin Angizi, Zhezhi He, Amro Awad, and Deliang Fan. MRIMA: An MRAM-based In-Memory Accelerator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(5):1123–1136, 2019.

[14] S. Angizi, A. S. Rakin, and D. Fan. CMP-PIM: An Energy-efficient Comparator-based Processing-in-Memory Neural Network Accelerator. In *DAC*, 2018.

[15] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology. In *MICRO*, 2017.

[16] Nastaran Hajinazar, Geraldo F Oliveira, Sven Gregorio, João Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gómez-Luna, and Onur Mutlu. SIMDRAM: A Framework for Bit-Serial SIMD Processing Using DRAM. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 329–345, 2021.

[17] Kaushik Roy, Indranil Chakraborty, Mustafa Ali, Aayush Ankit, and Amogh Agrawal. In-memory Computing In Emerging Memory Technologies for Machine Learning: An Overview. In *DAC*, 2020.

[18] Eduardo Perez, Cristian Zambelli, Mamathamba Kalishettyhalli Mahadevaiah, Piero Olivo, and Christian Wenger. Toward Reliable Multilevel Operation in RRAM Arrays: Improving Post-algorithm Stability and Assessing Endurance/Data Retention. *IEEE Journal of the Electron Devices Society*, 2019.

[19] Tim Daulby, Anand Savanth, Alex S Weddell, and Geoff V Merrett. Comparing NVM Technologies Through the Lens of Intermittent Computation. In *Proceedings of the 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, pages 77–78, 2020.

[20] Huai Lin, Xi Luo, Long Liu, Di Wang, Xuefeng Zhao, Ziwei Wang, Xiaoyong Xue, Feng Zhang, and Guozhong Xing. All-electrical Control of Compact SOT-MRAM: Toward Highly Efficient and Reliable Non-volatile In-memory Computing. *Micromachines*, 2022.

[21] Shimeng Yu and Pai-Yu Chen. Emerging Memory Technologies: Recent Trends and Prospects. *IEEE Solid-State Circuits Magazine*, 8(2):43–56, 2016.

[22] William J Gallagher, Eric Chien, Tien-Wei Chiang, Jian-Cheng Huang, Meng-Chun Shih, CY Wang, Christine Bair, George Lee, Yi-Chun Shih, Chia-Fu Lee, et al. Recent Progress and Next Directions for Embedded MRAM Technology. In *2019 Symposium on VLSI Circuits*, pages T190–T191. IEEE, 2019.

[23] Pranav Patel, Eamonn Keogh, Jessica Lin, and Stefano Lonardi. Mining Motifs in Massive Time Series Databases. In *ICDM*, 2002.

[24] Eamonn Keogh, Jessica Lin, Sang-Hee Lee, and Helga Van Herle. Finding the Most Unusual Time Series Subsequence: Algorithms and Applications. *Knowledge and Information Systems*, 11(1):1–27, 2007.

[25] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets. In *ICDM*, 2016.

[26] Chotirat Ann Ratanamahatana and Eamonn Keogh. Making Time-Series Classification More Accurate Using Learned Constraints. In *Proceedings of the 2004 SIAM international conference on data mining*, pages 11–22. SIAM, 2004.

[27] Robert H Shumway and David S Stoffer. Time Series Analysis and Its Applications: With R Examples. 2017.

[28] André E X Brown, Eviatar Yemini, Laura J Grundy, Tadas Jucikas, and William Schafer. A Dictionary of Behavioral Motifs Reveals Clusters of Genes Affecting Caenorhabditis Elegans Locomotion. *Proceedings of the National Academy of Sciences of the United States of America*, 110, 2012.

[29] Martin Straume. DNA Microarray Time Series Analysis: Automated Statistical Assessment of Circadian Rhythms in Gene Expression Patterning. In *Methods in Enzymology*, volume 383, pages 149–166. Elsevier, 2004.

[30] Ziv Bar-Joseph. Analyzing Time Series Gene Expression Data. *Bioinformatics*, 20(16):2493–2503, 2004.

[31] Arvind Balasubramanian, Jun Wang, and Balakrishnan Prabhakaran. Discovering Multidimensional Motifs in Physiological Signals for Personalized Healthcare. *JSTSP*, 2016.

[32] Yoshiki Tanaka, Kazuhisa Iwamoto, and Kuniaki Uehara. Discovery of Time-Series Motif from MultiDimensional Data Based on MDL Principle. *Machine Learning*, 58:269–300, 2005.

[33] Alireza Vahdatpour, Navid Amini, and Majid Sarrafzadeh. Toward Unsupervised Activity Discovery Using Multi-dimensional Motif Detection in Time Series. In *Int'l. Jont Conf. on Artifical Intelligence*, pages 1261–1266, 2009.

[34] Amy McGovern, Derek H. Rosendahl, Rodger A. Brown, and Kelvin K. Droegemeier. Identifying Predictive Multi-dimensional Time Series Motifs: an Application to Severe Weather Prediction. *Data Mining and Knowledge Discovery*, 22(1):232–258, 2011.

[35] Ilya Kolb, Giovanni Talei Franzesi, Michael Wang, Suhasa B. Kodandaramaiah, Craig R. Forest, Edward S. Boyden, and Annabelle C. Singer. Evidence for Long-Timescale Patterns of Synaptic Inputs in CA1 of Awake Behaving Mice. *Journal of Neuroscience*, 38(7):1821–1834, 2018.

[36] Balázs Szigeti, Ajinkya Deogade, and Barbara Webb. Searching for Motifs in the Behaviour of Larval Drosophila Melanogaster and Caenorhabditis Elegans Reveals Continuity Between Behavioural States. *Journal of The Royal Society Interface*, 12(113):20150899, 2015.

[37] T. Warren Liao. Clustering of Time Series Data - A Survey. *Pattern Recognition*, 38(11):1857–1874, 2005.

[38] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. Time-Series Clustering - A Decade Review. *Inf. Syst.*, 53(C):16–38, 2015.

[39] Eamonn Keogh and Shruti Kasetty. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. *Data Min. Knowl. Discov.*, 7(4):349–371, 2003.

[40] Daniel T. Trugman and Peter M Shearer. GrowClust: A Hierarchical Clustering Algorithm for Relative Earthquake Relocation, with Application to the Spanish Springs and Sheldon, Nevada, Earthquake Sequences. *Seismological Research Letters*, 88(2A), 2017.

[41] A Double-difference Earthquake Location Algorithm: Method and Application to the Northern Hayward Fault. *Bull. Seism. Soc. Am.*, 2000.

[42] Annemarie Christophersen, Natalia I. Deligne, Anca M. Hanea, Lauriane Chardot, Nicolas Fournier, and Willy P. Aspinall. Bayesian Network Modeling and Expert Elicitation for Probabilistic Eruption Forecasting: Pilot Study for Whakaari/White Island, New Zealand. *Frontiers in Earth Science*, 6:211, 2018.

[43] Chris McKee, Ima Itikarai, and Hugh 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: Volcano Crisis Communication*, pages 205–233. Springer International Publishing, 2018.

[44] E. Philip Howrey. The Role of Time Series Analysis in Econometric Model Evaluation. *Evaluation of Econometric Models*, pages 275–307, 1980.

[45] R.H. Shumway. Applied Statistical Time Series Analysis. *Prentice-Hall, Englewood Cliffs.*, 1988.

[46] Ruey S. Tsay. *Analysis of Financial Time Series*. Wiley Series in Probability and Statistics. Wiley-Interscience, 2005.

[47] Kei Nakagawa, Mitsuyoshi Imamura, and Kenichi Yoshida. Stock Price Prediction Using k-Medoids Clustering with Indexing Dynamic Time Warping. *Electronics and Communications in Japan*, 2019.

[48] George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley-Interscience, 2015.

[49] Alex T Nelson and Eric A Wan. A Two-Observation Kalman Framework for Maximum-Likelihood Modeling of Noisy Time Series. In *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36227)*, volume 3, pages 2489–2494. IEEE, 1998.

[50] David Barber, A Taylan Cemgil, and Silvia Chiappa. *Bayesian Time Series Models*. Cambridge University Press, 2011.

[51] S. A. P. Kumar and P. K. Bora. Time Series Analysis and Signal Processing. In *Conf. on Computational Intelligence and Signal Processing*, pages 24–24, 2012.

[52] Berlin Wu. Pattern Recognition and Classification in Time Series Analysis. *Applied Mathematics and Computation*, 62(1):29 – 45, 1994.

[53] Anukool Lakhina, Mark Crovella, and Christiphe Diot. Characterization of Network-wide Anomalies in Traffic Flows. In *IMC*, pages 201–206. ACM, 2004.

[54] Raj Jain and Shawn Routhier. Packet Trains–Measurements and a New Model for Computer Network Traffic. *IEEE Journal on Selected Areas in Communications*, 4(6):986–995, 1986.

[55] Paul Barford and David Plonka. Characteristics of Network Traffic Flow Anomalies. In *Internet Measurement Workshop*, pages 69–73. Citeseer, 2001.

[56] Lal Hussain, Wajid Aziz, Jalal S. Alowibdi, Nazneen Habib, Muhammad Rafique, Sharjil Saeed, and Syed Zaki Hassan Kazmi. Symbolic Time Series Analysis of Electroencephalographic (EEG) Epileptic Seizure and Brain Dynamics with Eye-open and Eye-closed Subjects During Resting States. *Journal of Physiological Anthropology*, 36(1), 2017.

[57] T. Balli and R. Palaniappan. EEG Time Series Analysis with Exponential Autoregressive Modelling. In *Canadian Conf. on Electrical and Computer Engineering*, 2008.

[58] Tim Dunn, Harisankar Sadasivan, Jack Wadden, Kush Goliya, Kuan-Yu Chen, David Blaauw, Reetuparna Das, and Satish Narayanasamy. SquiggleFilter: An Accelerator for Portable Virus Detection. In *MICRO*, pages 535–549, 2021.

[59] Guangyuan Chen, Guoliang Lu, Zhaohong Xie, and Wei Shang. Anomaly Detection in EEG Signals: a Case Study on Similarity Measure. *Computational Intelligence and Neuroscience*, 2020.

[60] R Vio, Niels Kristensen, Henrik Madsen, and W Wamsteker. Time Series Analysis in Astronomy: Limits and Potentialities. *Astronomy and Astrophysics*, 435, 10 2004.

[61] Jeffrey D. Scargle. Studies in Astronomical Time Series Analysis. V. Bayesian Blocks, a New Method to Analyze Structure in Photon Counting Data. *The Astrophysical Journal*, 504(1):405–418, 1998.

[62] Kajal Nusratullah, Shoab Ahmad Khan, Asadullah Shah, and Wasi Haider Butt. Detecting Changes in Context Using Time Series Analysis of Social Network. In *2015 SAI Intelligent Systems Conference (IntelliSys)*, pages 996–1001. IEEE, 2015.

[63] Sitaram Asur and Bernardo A Huberman. Predicting the Future with Social Media. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology-Volume 01*, pages 492–499. IEEE Computer Society, 2010.

[64] Anna Klos, Machiel S Bos, and Janusz Bogusz. Detecting Time-varying Seasonal Signal in GPS Position Time Series with Different Noise Levels. *GPS Solutions*, 22(1):1–11, 2018.

[65] Robert Stoermer, Ralph Mager, Andreas Roessler, Franz Mueller-Spahn, and Alex H Bullinger. Monitoring Human-virtual Reality Interaction: a Time Series Analysis Approach. *CyberPsychology & Behavior*, 3(3):401–406, 2000.

[66] Shah Muhammed Abid Hussain and ABM Harun-ur Rashid. User Independent Hand Gesture Recognition by Accelerated DTW. In *ICIEV*, pages 1033–1037. IEEE, 2012.

[67] Alina Delia Calin. Gesture Recognition on Kinect Time Series Data Using Dynamic Time Warping and Hidden Markov Models. In *SYNASC*, pages 264–271. IEEE, 2016.

[68] Samet Ayhan and Hanan Samet. Time Series Clustering of Weather Observations in Predicting Climb Phase of Aircraft Trajectories. In *IWCTS*, pages 25–30, 2016.

[69] Li Li, Xiaonan Su, Yi Zhang, Yuetong Lin, and Zhiheng Li. Trend Modeling for Traffic Time Series Analysis: An Integrated Study. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):3430–3439, 2015.

[70] Stephanie L Hyland, Martin Faltys, Matthias Hüser, Xinrui Lyu, Thomas Gumbsch, Cristóbal Esteban, Christian Bock, Max Horn, Michael Moor, Bastian Rieck, et al. Early Prediction of Circulatory Failure in the Intensive Care Unit Using Machine Learning. *Nature Medicine*, 26(3):364–373, 2020.

[71] Donald J Berndt and James Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, USA:, 1994.

[72] Renjie Wu and Eamonn J Keogh. FastDTW is Approximate and Generally Slower than the Algorithm it Approximates. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[73] Tuomas S Koskela, Mathieu Lobet, Jack Deslippe, and Zakhar Matveev. Roofline Analysis in the Intel® Advisor to Deliver Optimized Performance for applications on Intel® Xeon Phi™ Processor. Technical report, LBNL, Berkeley, 2017.

[74] Bertil Schmidt and Christian Hundt. cuDTW++: Ultra-Fast Dynamic Time Warping on CUDA-Enabled GPUs. In *Euro-Par*, pages 597–612. Springer International Publishing, 2020.

[75] NVIDIA Visual Profiler. https://developer.nvidia.com/nvidia-visual-profiler. Accessed 16 November 2023.

[76] UPMEM. Introduction to UPMEM PIM. Processing-in-Memory (PIM) on DRAM Accelerator (White Paper), 2018.

[77] Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. RAIDR: Retention-aware Intelligent DRAM Refresh. *ACM SIGARCH Computer Architecture News*, 2012.

[78] Pengfei Zuo, Yu Hua, Ming Zhao, Wen Zhou, and Yuncheng Guo. Improving the Performance and Enndurance of Encrypted Non-Volatile Main Memory Through Deduplicating Writes. In *MICRO*, 2018.

[79] Alessandro Grossi, Cristian Zambelli, Piero Olivo, Paolo Pellati, Michele Ramponi, Christian Wenger, Jeremy Alvarez-Herault, and Ken Mackay. An Automated Test Equipment for Characterization of Emerging MRAM and RRAM Arrays. *IEEE Transactions on Emerging Topics in Computing*, 6(2):269–277, 2016.

[80] Sparsh Mittal. A Survey of ReRAM-based Architectures for Processing-In-Memory and Neural Networks. *Machine Learning and Knowledge Extraction*, 1(1):75–114, 2018.

[81] Yue Zhang, Jinkai Wang, Chenyu Lian, Yining Bai, Guanda Wang, Zhizhong Zhang, Zhenyi Zheng, Lei Chen, Kun Zhang, Georgios Sirakoulis, et al. Time-domain Computing in Memory Using Spintronics for Energy-efficient Convolutional Neural Network. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(3):1193–1205, 2021.

[82] Xing Jin, Weichong Chen, Ximing Li, Ningyuan Yin, Caihua Wan, Mingkun Zhao, Xiufeng Han, and Zhiyi Yu. High-reliability, Reconfigurable, and Fully Non-volatile Full-adder Based on SOT-MTJ for Image Processing Applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 70(2):781–785, 2022.

[83] Jinkai Wang, Yining Bai, Hongyu Wang, Zuolei Hao, Guanda Wang, Kun Zhang, Youguang Zhang, Weifeng Lv, and Yue Zhang. Reconfigurable Bit-serial Operation Using Toggle SOT-MRAM for High-performance Computing in Memory Architecture. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 69(11):4535–4545, 2022.

[84] Shahar Kvatinsky, Dmitry Belousov, Slavik Liman, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. MAGIC—Memristor-aided Logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, 2014.

[85] Antonio J Dios, Angeles Navarro, Rafael Asenjo, Francisco Corbera, and Emilio L Zapata. A Case Study of the Task-based Parallel Wavefront Pattern. In *Applications, Tools and Techniques on the Road to Exascale Computing*, pages 65–72. IOS Press, 2012.

[86] ZSim+Ramulator. github.com/CMU-SAFARI/ramulator-pim, 2022.

[87] Rajesh Saha, Yogendra Pratap Pundir, and Pankaj Kumar Pal. Comparative Analysis of STT and SOT Based MRAMs for Last Level Caches. *Journal of Magnetism and Magnetic Materials*, 551:169161, 2022.

[88] Tetsuo Endoh, Hiroaki Honjo, Koichi Nishioka, and Shoji Ikeda. Recent Progresses in STT-MRAM and SOT-MRAM for next generation MRAM. In *2020 IEEE Symposium on VLSI Technology*, pages 1–2. IEEE, 2020.

[89] Ashok Veeraraghavan, Rama Chellappa, and Amit K Roy-Chowdhury. The Function Space of an Activity. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 959–968. IEEE, 2006.

[90] C M Yeh, H V Herle, and E Keogh. Matrix Profile III: The Matrix Profile Allows Visualization of Salient Subsequences in Massive Time Series. In *ICDM*, 2016.

[91] Penguin Data. https://www.cs.ucr.edu/~eamonn/MatrixProfile.html, 2022.

[92] David Murray, Jing Liao, Lina Stankovic, Vladimir Stankovic, Richard Hauxwell-Baldwin, Charlie Wilson, Michael Coleman, Tom Kane, and Steven Firth. A Data Management Platform for Personalised Real-Time Energy Feedback. 2015.

[93] A. Taddei, G. Distante, M. Emdin, P. Pisani, G. B. Moody, C. Zeelenberg, and C. Marchesi. The European ST-T Database: Standard for Evaluating Systems for the Analysis of ST-T Changes in Ambulatory Electrocardiography. *European Heart Journal*, 1992.

[94] Mesbah Uddin and Garrett S Rose. A Practical Sense Amplifier Design for Memristive Crossbar Circuits (PUF). In *2018 31st IEEE International System-on-Chip Conference (SOCC)*, pages 209–214. IEEE, 2018.

[95] Yeongkyo Seo, Kon-Woo Kwon, and Kaushik Roy. Area-efficient SOT-MRAM with a Schottky Diode. *IEEE Electron Device Letters*, 37(8):982–985, 2016.

[96] Harisankar Sadasivan, Daniel Stiffler, Ajay Tirumala, Johnny Israeli, and Satish Narayanasamy. Accelerated Dynamic Time Warping on GPU for Selective Nanopore Sequencing. *bioRxiv*, 2023.

[97] Seongyoung Kang, Jinyeong Moon, and Sang-Woo Jun. FPGA-Accelerated Time Series Mining on Low-Power IoT Devices. In *ASAP*, 2020.

[98] Zilong Wang, Sitao Huang, Lanjun Wang, Hao Li, Yu Wang, and Huazhong Yang. Accelerating Subsequence Similarity Search Based on Dynamic Time Warping Distance with FPGA. FPGA, 2013.

[99] Saugata Ghose, Amirali Boroumand, Jeremie S Kim, Juan Gómez-Luna, and Onur Mutlu. Processing-in-Memory: A Workload-driven Perspective. *IBM JRD*, 2019.

[100] Damla Senol Cali, Gurpreet S Kalsi, Zülal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, et al. GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis. In *MICRO*, 2020.

[101] Gagandeep Singh, Juan Gómez-Luna, Giovanni Mariani, Geraldo F Oliveira, Stefano Corda, Sander Stuijk, Onur Mutlu, and Henk Corporaal. NAPEL: Near-memory Computing Application Performance Prediction Via Ensemble Learning. In *DAC*, 2019.

[102] Gagandeep Singh, Dionysios Diamantopoulos, Christoph Hagleitner, Juan Gomez-Luna, Sander Stuijk, Onur Mutlu, and Henk Corporaal. NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling. In *FPL*, 2020.

[103] Harold S Stone. A Logic-in-Memory Computer. *IEEE TC*, 1970.

[104] W. H. Kautz. Cellular Logic-in-Memory Arrays. *IEEE TC*, 1969.

[105] David Elliot Shaw, Salvatore J. Stolfo, Hussein Ibrahim, Bruce Hillyer, Gio Wiederhold, and JA Andrews. The NON-VON Database Machine: A Brief Overview. *IEEE Database Eng. Bull.*, 1981.

[106] P. M. Kogge. EXECUBE - A New Architecture for Scaleable MPPs. In *ICPP*, 1994.

[107] Maya Gokhale, Bill Holmes, and Ken Iobst. Processing in Memory: The Terasys Massively Parallel PIM Array. *IEEE Computer*, 1995.

[108] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. A Case for Intelligent RAM. *IEEE Micro*, 1997.

[109] Mark Oskin, Frederic T. Chong, and Timothy Sherwood. Active Pages: A Computation Model for Intelligent Memory. In *ISCA*, 1998.

[110] Yi Kang, Wei Huang, Seung-Moon Yoo, Diana Keen, Zhenzhou Ge, Vinh Lam, Pratap Pattnaik, and Josep Torrellas. FlexRAM: Toward an Advanced Intelligent Memory System. In *ICCD*, 1999.

[111] Ken Mai, Tim Paaske, Nuwan Jayasena, Ron Ho, William J. Dally, and Mark Horowitz. Smart Memories: A Modular Reconfigurable Architecture. In *ISCA*, 2000.

[112] Richard C Murphy, Peter M Kogge, and Arun Rodrigues. The Characterization of Data Intensive Memory Workloads on Distributed PIM Systems. In *Intelligent Memory Systems*. Springer, 2001.

[113] Jeff Draper, Jacqueline Chame, Mary Hall, Craig Steele, Tim Barrett, Jeff LaCoss, John Granacki, Jaewook Shin, Chun Chen, Chang Woo Kang, Ihn Kim, and Gokhan Daglikoca. The Architecture of the DIVA Processing-in-Memory Chip. In *SC*, 2002.

[114] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna Das. Compute Caches. In *HPCA*, 2017.

[115] Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaaauw, and Reetuparna Das. Neural Cache: Bit-serial In-cache Acceleration of Deep Neural Networks. In *ISCA*, 2018.

[116] Daichi Fujiki, Scott Mahlke, and Reetuparna Das. Duality Cache for Data Parallel Acceleration. In *ISCA*, 2019.

[117] Mingu Kang, Min-Sun Keel, Naresh R Shanbhag, Sean Eilert, and Ken Curewitz. An Energy-Efficient VLSI Architecture for Pattern Recognition via Deep Embedding of Computation in SRAM. In *ICASSP*, 2014.

[118] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry. Buddy-RAM: Improving the Performance and Efficiency of Bulk Bitwise Operations Using DRAM. arXiv:1611.09988 [cs:AR], 2016.

[119] Vivek Seshadri, K. Hsieh, A. Boroumand, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry. Fast Bulk Bitwise AND and OR in DRAM. *CAL*, 2015.

[120] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Michael A Kozuch, Phillip B Gibbons, and Todd C. Mowry. RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization. In *MICRO*, 2013.

[121] Shaahin Angizi and Deliang Fan. Graphide: A Graph Processing Accelerator Leveraging In-DRAM-computing. In *GLSVLSI*, 2019.

[122] J. Kim, M. Patel, H. Hassan, and O. Mutlu. The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern DRAM Devices. In *HPCA*, 2018.

[123] J. Kim, M. Patel, H. Hassan, L. Orosa, and O. Mutlu. D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput. In *HPCA*, 2019.

[124] Fei Gao, Georgios Tziantzioulis, and David Wentzlaff. ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs. In *MICRO*, 2019.

[125] Kevin K. Chang, Prashant J. Nair, Donghyuk Lee, Saugata Ghose, Moinuddin K. Qureshi, and Onur Mutlu. Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM. In *HPCA*, 2016.

[126] Xin Xin, Youtao Zhang, and Jun Yang. ELP2IM: Efficient and Low Power Bitwise Operation Processing in DRAM. In *HPCA*, 2020.

[127] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang. DrAcc: A DRAM Based Accelerator for Accurate CNN Inference. In *DAC*, 2018.

[128] S. H. S. Rezaei, M. Modarressi, R. Ausavarungnirun, M. Sadrosadati, O. Mutlu, and M. Daneshtalab. NoM: Network-on-Memory for Inter-Bank Data Transfer in Highly-Banked Memories. *CAL*, 2020.

[129] Yaohua Wang, Lois Orosa, Xiangjun Peng, Yang Guo, Saugata Ghose, Minesh Patel, Jeremie S Kim, Juan Gómez Luna, Mohammad Sadrosadati, Nika Mansouri Ghiasi, et al. FIGARO: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching. In *MICRO*, 2020.

[130] Mustafa F Ali, Akhilesh Jaiswal, and Kaushik Roy. In-Memory Low-Cost Bit-Serial Addition Using Commodity DRAM Technology. In *TCAS-I*, 2019.

[131] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-Volatile Memories. In *DAC*, 2016.

[132] S. Angizi, Z. He, and D. Fan. PIMA-Logic: A Novel Processing-in-Memory Architecture for Highly Flexible and Energy-efficient Logic Computation. In *DAC*, 2018.

[133] S. Angizi, J. Sun, W. Zhang, and D. Fan. AlignS: A Processing-in-Memory Accelerator for DNA Short Read Alignment Leveraging SOT-MRAM. In *DAC*, 2019.

[134] Yifat Levy, Jehoshua Bruck, Yuval Cassuto, Eby G. Friedman, Avinoam Kolodny, Eitan Yaakobi, and Shahar Kvatinsky. Logic Operations in Memory Using a Memristive Akers Array. *Microelectronics Journal*, 2014.

[135] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. ISAAC: A Convolutional Neural Network Accelerator with In-situ Analog Arithmetic in Crossbars. *ISCA*, 2016.

[136] S. Kvatinsky, A. Kolodny, U. C. Weiser, and E. G. Friedman. Memristor-Based IMPLY Logic Design Procedure. In *ICCD*, 2011.

[137] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser. Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies. *TVLSI*, 2014.

[138] P.-E. Gaillardon, L. Amaru, A. Siemon, and et al. The Programmable Logic-in-Memory (PLiM) Computer. In *DATE*, 2016.

[139] D. Bhattacharjee, R. Devadoss, and A. Chattopadhyay. ReVAMP: ReRAM based VLIW Architecture for In-memory Computing. In *DATE*, 2017.

[140] S. Hamdioui, L. Xie, H. A. D. Nguyen, and et al. Memristor Based Computation-in-Memory Architecture for Data-intensive Applications. In *DATE*, 2015.

[141] L. Xie, H. A. D. Nguyen, M. Taouil, and et al. Fast Boolean Logic Papped on Memristor Crossbar. In *ICCD*, 2015.

[142] S. Hamdioui, S. Kvatinsky, and et al. G. Cauwenberghs. Memristor for Computing: Myth or Reality? In *DATE*, 2017.

[143] J. Yu, H. A. D. Nguyen, L. Xie, and et al. Memristive Devices for Computation-in-Memory. In *DATE*, 2018.

[144] Christina Giannoula, Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas, Ivan Fernandez, Juan Gómez-Luna, Lois Orosa, Nectarios Koziris, Georgios Goumas, and Onur Mutlu. SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures. In *HPCA*, pages 263–276. IEEE, 2021.

[145] Ivan Fernandez, Ricardo Quislant, Eladio Gutiérrez, Oscar Plata, Christina Giannoula, Mohammed Alser, Juan Gómez-Luna, and Onur Mutlu. NATSA: A Near-data Processing Accelerator for Time Series Analysis. In *ICCD*, 2020.

[146] J. S. Kim, D. Senol, 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*, 2018.

[147] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture. In *ISCA*, 2015.

[148] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing. In *ISCA*, 2015.

[149] 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 *ASPLOS*, 2018.

[150] Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh, Nastaran Hajinazar, Krishna T Malladi, Hongzhong Zheng, et al. CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators. In *ISCA*, 2019.

[151] Hadi Asghari-Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim. Chameleon: Versatile and Practical Near-DRAM Acceleration Architecture for Large Memory Systems. In *MICRO*, 2016.

[152] Oreoluwatomiwa O. Babarinsa and Stratos Idreos. JAFAR: Near-Data Processing for Databases. In *SIGMOD*, 2015.

[153] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie. PRIME: A Novel Processing-In-Memory Architecture for Neural Network Computation In ReRAM-Based Main Memory. In *ISCA*, 2016.

[154] Amin Farmahini-Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules. In *HPCA*, 2015.

[155] Mingyu Gao, Grant Ayers, and Christos Kozyrakis. Practical Near-Data Processing for In-Memory Analytics Frameworks. In *PACT*, 2015.

[156] Mingyu Gao and Christos Kozyrakis. HRL: Efficient and Flexible Reconfigurable Logic for Near-Data Processing. In *HPCA*, 2016.

[157] Boncheol 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 *ISCA*, 2016.

[158] Q. Guo, N. Alachiotis, B. Akin, F. Sadi, G. Xu, T. M. Low, L. Pileggi, J. C. Hoe, and F. Franchetti. 3D-Stacked Memory-Side Acceleration: Accelerator and System Design. In *WoNDP*, 2014.

[159] Milad Hashemi, Khubaib, Eiman Ebrahimi, Onur Mutlu, and Yale N. Patt. Accelerating Dependent Cache Misses with an Enhanced Memory Controller. In *ISCA*, 2016.

[160] M. Hashemi, O. Mutlu, and Y. N. Patt. Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads. In *MICRO*, 2016.

[161] Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Conner, Nandita Vijaykumar, Onur Mutlu, and Stephen Keckler. Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems. In *ISCA*, 2016.

[162] Duckhwan Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay. Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory. In *ISCA*, 2016.

[163] G. Kim, N. Chatterjee, M. O'Connor, and K. Hsieh. Toward Standardized Near-Data Processing with Unrestricted Data Placement for GPUs. In *SC*, 2017.

[164] Joo Hwan Lee, Jaewoong Sim, and Hyesoon Kim. BSSync: Processing Near Memory for Machine Learning Workloads with Bounded Staleness Consistency Models. In *PACT*, 2015.

[165] Zhiyu Liu, Irina Calciu, Maurice Herlihy, and Onur Mutlu. Concurrent Data Structures for Near-Memory Computing. In *SPAA*, 2017.

[166] Amir Morad, Leonid Yavits, and Ran Ginosar. GP-SIMD Processing-in-Memory. *ACM TACO*, 2015.

[167] Lifeng Nai, Ramyad Hadidi, Jaewoong Sim, Hyojong Kim, Pranith Kumar, and Hyesoon Kim. GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks. In *HPCA*, 2017.

[168] Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayiran, Asit K. Mishra, Mahmut T. Kandemir, Onur Mutlu, and Chita R. Das. Scheduling Techniques for GPU Architectures with Processing-in-Memory Capabilities. In *PACT*, 2016.

[169] Seth H. Pugsley, Jeffrey Jestes, Huihui Zhang, Rajeev Balasubramonian, Vijayalakshmi Srinivasan, Alper Buyuktosunoglu, Al Davis, and Feifei Li. NDC: Analyzing the Impact of 3D-Stacked Memory+Logic Devices on MapReduce Workloads. In *ISPASS*, 2014.

[170] D. P. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski. TOP-PIM: Throughput-Oriented Programmable Processing in Memory. In *HPDC*, 2014.

[171] Qiuling Zhu, Tobias Graf, H Ekin Sumbul, Larry Pileggi, and Franz Franchetti. Accelerating Sparse Matrix-Matrix Multiplication with 3D-Stacked Logic-in-Memory Hardware. In *HPEC*, 2013.

[172] Berkin Akin, Franz Franchetti, and James C. Hoe. Data Reorganization in Memory Using 3D-Stacked DRAM. In *ISCA*, 2015.

[173] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. Tetris: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *ASPLOS*, 2017.

[174] Mario Drumond, Alexandros Daglis, Nooshin Mirzadeh, Dmitrii Ustiugov, Javier Picorel Obando, Babak Falsafi, Boris Grot, and Dionisios Pnevmatikatos. The Mondrian Data Engine. In *ISCA*, 2017.

[175] Guohao Dai, Tianhao Huang, Yuze Chi, Jishen Zhao, Guangyu Sun, Yongpan Liu, Yu Wang, Yuan Xie, and Huazhong Yang. GraphH: A Processing-in-Memory Architecture for Large-scale Graph Processing. *IEEE TCAD*, 2018.

[176] Mingxing Zhang, Youwei Zhuo, Chao Wang, Mingyu Gao, Yongwei Wu, Kang Chen, Christos Kozyrakis, and Xuehai Qian. GraphP: Reducing Communication for PIM-based Graph Processing with Efficient Data Partition. In *HPCA*, 2018.

[177] Yu Huang, Long Zheng, Pengcheng Yao, Jieshan Zhao, Xiaofei Liao, Hai Jin, and Jingling Xue. A Heterogeneous PIM Hardware-Software Co-Design for Energy-Efficient Graph Processing. In *IPDPS*, 2020.

[178] Youwei Zhuo, Chao Wang, Mingxing Zhang, Rui Wang, Dimin Niu, Yanzhi Wang, and Xuehai Qian. GraphQ: Scalable PIM-based Graph Processing. In *MICRO*, 2019.

[179] Paulo C Santos, Geraldo F Oliveira, Diego G Tomé, Marco AZ Alves, Eduardo C Almeida, and Luigi Carro. Operand Size Reconfiguration for Big Data Processing in Memory. In *DATE*, 2017.

[180] Saugata Ghose, Amirali Boroumand, Jeremie S Kim, Juan Gómez-Luna, and Onur Mutlu. Processing-in-Memory: A Workload-Driven Perspective. *IBM JRD*, 2019.

[181] Wen-Mei Hwu, Izzat El Hajj, Simon Garcia De Gonzalo, Carl Pearson, Nam Sung Kim, Deming Chen, Jinjun Xiong, and Zehra Sura. Rebooting the Data Access Hierarchy of Computing Systems. In *ICRC*, 2017.

[182] Maciej Besta, Raghavendra Kanakagiri, Grzegorz Kwasniewski, Rachata Ausavarungnirun, Jakub Beránek, Konstantinos Kanellopoulos, Kacper Janda, Zur Vonarburg-Shmaria, Lukas Gianinazzi, Ioana Stefan, et al. SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems. In *MICRO*, 2021.

[183] João Dinis Ferreira, Gabriel Falcao, Juan Gómez-Luna, Mohammed Alser, Lois Orosa, Mohammad Sadrosadati, Jeremie S. Kim, Geraldo F. Oliveira, Taha Shahroodi, Anant Nori, and Onur Mutlu. pLUTo: In-DRAM Lookup Tables to Enable Massively Parallel General-Purpose Computation. *arXiv:2104.07699 [cs.AR]*, 2021.

[184] Ataberk Olgun, Minesh Patel, Abdullah Giray Yağlıkçı, Haocong Luo, Jeremie S. Kim, F. Nisa Bostancı, Nandita Vijaykumar, Oğuz Ergin, and Onur Mutlu. QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAMs. In *ISCA*, 2021.

[185] Scott Lloyd and Maya Gokhale. In-memory Data Rearrangement for Irregular, Data-intensive Computing. *Computer*, 2015.

[186] Duncan G Elliott, Michael Stumm, W Martin Snelgrove, Christian Cojocaru, and Robert McKenzie. Computational RAM: Implementing Processors in Memory. *IEEE Design & Test of Computers*, 1999.

[187] Le Zheng, Sangho Shin, Scott Lloyd, Maya Gokhale, Kyungmin Kim, and Sung-Mo Kang. RRAM-based TCAMs for Pattern Search. In *ISCAS*, 2016.

[188] Joshua Landgraf, Scott Lloyd, and Maya Gokhale. Combining Emulation and Simulation to Evaluate a Near Memory Key/Value Lookup Accelerator, 2021.

[189] Arun Rodrigues, Maya Gokhale, and Gwendolyn Voskuilen. Towards a Scatter-Gather Architecture: Hardware and Software Issues. In *MEMSYS*, 2019.

[190] Scott Lloyd and Maya Gokhale. Design Space Exploration of Near Memory Accelerators. In *MEMSYS*, 2018.

[191] Scott Lloyd and Maya Gokhale. Near Memory Key/Value Lookup Acceleration. In *MEMSYS*, 2017.

[192] Maya Gokhale, Scott Lloyd, and Chris Hajas. Near Memory Data Structure Rearrangement. In *MEMSYS*, 2015.

[193] Ravi Nair, Samuel F Antao, Carlo Bertolli, Pradip Bose, Jose R Brunheroto, Tong Chen, C-Y Cher, Carlos HA Costa, Jun Doi, Constantinos Evangelinos, et al. Active Memory Cube: A Processing-in-Memory Architecture for Exascale Systems. *IBM JRD*, 2015.

[194] Arpith C Jacob, Zehra Sura, Tong Chen, Carlo Bertolli, Samuel Antao, Olivier Sallenave, Kevin O'Brien, Hans Jacobson, Ravi Nair, Jose R Brunheroto, et al. Compiling for the Active Memory Cube. Technical report, Tech. rep. RC25644 (WAT1612-008). IBM Research Division, 2016.

[195] Zehra Sura, Arpith Jacob, Tong Chen, Bryan Rosenburg, Olivier Sallenave, Carlo Bertolli, Samuel Antao, Jose Brunheroto, Yoonho Park, Kevin O'Brien, et al. Data Access Optimization in a Processing-in-Memory System. In *CF*, 2015.

[196] Ravi Nair. Evolution of Memory Architecture. *Proceedings of the IEEE*, 2015.

[197] Rajeev Balasubramonian, Jichuan Chang, Troy Manning, Jaime H Moreno, Richard Murphy, Ravi Nair, and Steven Swanson. Near-Data Processing: Insights from a MICRO-46 Workshop. *IEEE Micro*, 2014.

[198] Yue Xi, Bin Gao, Jianshi Tang, An Chen, Meng-Fan Chang, Xiaobo Sharon Hu, Jan Van Der Spiegel, He Qian, and Huaqiang Wu. In-Memory Learning With Analog Resistive Switching Memory: A Review and Perspective. *Proceedings of the IEEE*, 2020.

[199] 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 *ICCD*, 2016.

[200] Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu. LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory. *CAL*, 2016.

[201] Christina Giannoula, Ivan Fernandez, Juan Gómez-Luna, Nectarios Koziris, Georgios Goumas, and Onur Mutlu. Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-in-Memory Architectures. In *SIGMETRICS*, 2022.

[202] Alain Denzler, Rahul Bera, Nastaran Hajinazar, Gagandeep Singh, Geraldo F Oliveira, Juan Gómez-Luna, and Onur Mutlu. Casper: Accelerating Stencil Computation Using Near-cache Processing. *arXiv preprint arXiv:2112.14216*, 2021.

[203] Amirali Boroumand, Saugata Ghose, Geraldo F Oliveira, and Onur Mutlu. Polynesia: Enabling Effective Hybrid Transactional/Analytical Databases with Specialized Hardware/Software Co-Design. *arXiv:2103.00798 [cs.AR]*, 2021.

[204] Amirali Boroumand, Saugata Ghose, Geraldo F Oliveira, and Onur Mutlu. Polynesia: Enabling Effective Hybrid Transactional Analytical Databases with Specialized Hardware Software Co-Design. In *ICDE*, 2022.

[205] Gagandeep Singh, Mohammed Alser, Damla Senol Cali, Dionysios Diamantopoulos, Juan Gómez-Luna, Henk Corporaal, and Onur Mutlu. FPGA-based Near-Memory Acceleration of Modern Data-Intensive Applications. *IEEE Micro*, 2021.

[206] Gagandeep Singh, Dionysios Diamantopoulos, Juan Gómez-Luna, Christoph Hagleitner, Sander Stuijk, Henk Corporaal, and Onur Mutlu. Accelerating Weather Prediction using Near-Memory Reconfigurable Fabric. *ACM TRETS*, 2021.

[207] Jose M Herruzo, Ivan Fernandez, Sonia González-Navarro, and Oscar Plata. Enabling Fast and Energy-Efficient FM-Index Exact Matching Using Processing-Near-Memory. *The Journal of Supercomputing*, 2021.

[208] Leonid Yavits, Roman Kaplan, and Ran Ginosar. GIRAF: General Purpose In-Storage Resistive Associative Framework. *IEEE TPDS*, 2021.

[209] Bahar Asgari, Ramyad Hadidi, Jiashen Cao, Da Eun Shim, Sung-Kyu Lim, and Hyesoon Kim. FAFNIR: Accelerating Sparse Gathering by Using Efficient Near-Memory Intelligent Reduction. In *HPCA*, 2021.

[210] Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F. Oliveira, Xiaoyu Ma, Eric Shiu, and Onur Mutlu. Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks. In *PACT*, 2021.

[211] Amirali Boroumand. *Practical Mechanisms for Reducing Processor-Memory Data Movement in Modern Workloads*. PhD thesis, Carnegie Mellon University, 2020.

[212] Vivek Seshadri and Onur Mutlu. Simple Operations in Memory to Reduce Data Movement. In *Advances in Computers, Volume 106*. 2017.

[213] Safaa Diab, Amir Nassereldine, Mohammed Alser, Juan Gómez Luna, Onur Mutlu, and Izzat El Hajj. High-throughput Pairwise Alignment with the Wavefront Algorithm using Processing-in-Memory. In *HICOMB*, 2022.

[214] Daichi Fujiki, Scott Mahlke, and Reetuparna Das. In-Memory Data Parallel Processor. In *ASPLOS*, 2018.

[215] Yue Zha and Jing Li. Hyper-AP: Enhancing Associative Processing Through A Full-Stack Optimization. In *ISCA*, 2020.

[216] Onur Mutlu. Memory Scaling: A Systems Architecture Perspective. *IMW*, 2013.

[217] Onur Mutlu and Lavanya Subramanian. Research Problems and Opportunities in Memory Systems. *SUPERFRI*, 2014.

[218] Hameeza Ahmed, Paulo C Santos, João PC Lima, Rafael F Moura, Marco AZ Alves, Antônio CS Beck, and Luigi Carro. A Compiler for Automatic Selection of Suitable Processing-in-Memory Instructions. In *DATE*, 2019.

[219] Shubham Jain, Sachin Sapatnekar, Jian-Ping Wang, Kaushik Roy, and Anand Raghunathan. Computing-in-Memory With Spintronics. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1640–1645. IEEE, 2018.

[220] Nika Mansouri Ghiasi, Jisung Park, Harun Mustafa, Jeremie Kim, Ataberk Olgun, Arvid Gollwitzer, Damla Senol Cali, Can Firtina, Haiyu Mao, Nour Almadhoun Alserr, et al. GenStore: A High-Performance and Energy-Efficient In-Storage Computing System for Genome Sequence Analysis. In *ASPLOS*, 2022.

[221] Geraldo F. Oliveira, Juan Gómez-Luna, Lois Orosa, Saugata Ghose, Nandita Vijaykumar, Ivan Fernandez, Mohammad Sadrosadati, and Onur Mutlu. DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks. *IEEE Access*, 2021.

[222] Seunghwan Cho, Haerang Choi, Eunhyeok Park, Hyunsung Shin, and Sungjoo Yoo. McDRAM v2: In-Dynamic Random Access Memory Systolic Array Accelerator to Address the Large Model Problem in Deep Neural Networks on the Edge. *IEEE Access*, 2020.

[223] Hyunsung Shin, Dongyoung Kim, Eunhyeok Park, Sungho Park, Yongsik Park, and Sungjoo Yoo. McDRAM: Low Latency and Energy-efficient Matrix Computations in DRAM. *IEEE TCADICS*, 2018.

[224] Peng Gu, Xinfeng Xie, Yufei Ding, Guoyang Chen, Weifeng Zhang, Dimin Niu, and Yuan Xie. iPIM: Programmable In-Memory Image Processing Accelerator using Near-Bank Architecture. In *ISCA*, 2020.

[225] D. Lavenier, R. Cimadomo, and R. Jodin. Variant Calling Parallelization on Processor-in-Memory Architecture. In *BIBM*, 2020.

[226] Vasileios Zois, Divya Gupta, Vassilis J. Tsotras, Walid A. Najjar, and Jean-Francois Roy. Massively Parallel Skyline Computation for Processing-in-Memory Architectures. In *PACT*, 2018.
[227] UPMEM. UPMEM Website. https://www.upmem.com, 2023.
[228] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System. *IEEE Access*, 2022.
[229] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. Benchmarking Memory-Centric Computing Systems: Analysis of Real Processing-In-Memory Hardware. In *IGSC*, 2021.
[230] Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu. Accelerating Pointer Chasing in 3D-stacked Memory: Challenges, Mechanisms, Evaluation. In *ICCD*, 2016.
[231] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A Scalable Processing-In-Memory Accelerator for Parallel Graph Processing. In *ISCA*, 2015.
[232] V T Lee, A Mazumdar, C C del Mundo, A Alaghi, L Ceze, and M Oskin. Application Codesign of NDP for Similarity Search. In *IPDPS*, 2018.
[233] Hongyi Xin, Donghyuk Lee, Farhad Hormozdiari, Can Alkan, and Onur Mutlu. FastHASH: A New GPU-friendly Algorithm for Fast and Comprehensive Next-Generation Sequence Mapping. In *BMC Genomics*, 2013.
[234] Mohammed Alser, Hasan Hassan, Akash Kumar, Onur Mutlu, and Can Alkan. Shouji: a Fast and Efficient Pre-alignment Filter for Sequence Alignment. *Bioinformatics*, 2019.
[235] Hongyi Xin, John Greth, John Emmons, Gennady Pekhimenko, Carl Kingsford, Can Alkan, and Onur Mutlu. Shifted Hamming Distance: A Fast and Accurate SIMD-friendly Filter to Accelerate Alignment Verification in Read Mapping. *Bioinformatics*, 2015.
[236] Mohammed Alser, Taha Shahroodi, Juan Gomez-Luna, Can Alkan, and Onur Mutlu. SneakySnake: A Fast and Accurate Universal Genome Pre-Alignment Filter for CPUs, GPUs, and FPGAs. *arXiv*, 2019.
[237] Mohammed Alser, Hasan Hassan, Hongyi Xin, Oğuz Ergin, Onur Mutlu, and Can Alkan. GateKeeper: A New Hardware Arch. for Accelerating Pre-alignment in DNA Short Read Mapping. *Bioinformatics*, 2017.
[238] Mohammed Alser, Zülal Bingöl, Damla Senol Cali, Jeremie Kim, Saugata Ghose, Can Alkan, and Onur Mutlu. Accelerating Genome Analysis: A Primer on an Ongoing Journey. *IEEE Micro*, 2020.
[239] Christina Giannoula, Ivan Fernandez, Juan Gómez-Luna, Nectarios Koziris, Georgios Goumas, and Onur Mutlu. SparseP: Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-In-Memory Architectures. In *Proc. ACM Meas. Anal. Comput. Syst.*, 2022.
[240] Christina Giannoula, Ivan Fernandez, Juan Gómez-Luna, Nectarios Koziris, Georgios Goumas, and Onur Mutlu. Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-In-Memory Architectures. In *SIGMETRICS*, 2022.
[241] Xiaowei Xu, Feng Lin, Aosen Wang, Xinwei Yao, Qing Lu, Wenyao Xu, Yiyu Shi, and Yu Hu. Accelerating Dynamic Time Warping With Memristor-Based Customized Fabrics. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 37(4):729–741, 2018.
[242] Zhengyu Chen and Jie Gu. High-Throughput Dynamic Time Warping Accelerator for Time-Series Classification With Pipelined Mixed-Signal Time-Domain Computing. *IEEE Journal of Solid-State Circuits*, 56(2):624–635, 2021.