# A Study of STT-RAM-based In-Memory Computing Across the Memory Hierarchy

Dhruv Gajaria, Kevin Antony Gomez, and Tosiron Adegbija
Department of Electrical & Computer Engineering
University of Arizona, Tucson, AZ, USA
Email: {dhruvgajaria, kevingomez, tosiron}@arizona.edu

***Abstract*—In-memory computing (or processing in memory) is a promising approach to reducing the data transfer bottleneck in computer systems by bringing computation closer to the memory. Prior work proposed using *Spin-Transfer Torque RAM (STT-RAM)* for in-memory computing to leverage STT-RAM's numerous advantages, including non-volatility, near-zero leakage power, high area density, better endurance than other non-volatile memory technologies and demonstrated commercial viability. This paper explores, for the first time, the tradeoffs of STT-RAM in-memory computing across the memory hierarchy, including the main memory and cache hierarchy. We explore a system model in which processing in memory (PiM) occurs in non-volatile STT-RAM, whereas processing in cache (PiC) occurs in relaxed retention (volatile) STT-RAM. In relaxed retention STT-RAM caches, the retention time—the duration for which the STT-RAM cell retains data—is significantly reduced to mitigate STT-RAM's intrinsic write latency and write energy overheads. Importantly, we also analyze the tradeoffs and overheads of data movement for PiC vs. write overheads for PiM for STT-RAMs. The analysis is performed in the context of different kinds of workloads to explore the impacts of various workload characteristics (e.g., temporal locality, computational intensity, CPU-dependent workloads with limited instruction-level parallelism) on PiC/PiM tradeoffs. Using these workloads, we also evaluate computing in STT-RAM vs. SRAM at different levels of the cache hierarchy. Our analysis reveals that STT-RAM-based PiC has promising advantages over PiM in certain workload contexts and offers solutions to some of the challenges that arise in implementing PiC-enabled systems.**

***Index Terms*—STT-RAM, relaxed retention time, in-memory computing, in-cache computing.**

## I. Introduction

The growth of data-intensive applications has given rise to data bottlenecks in computer systems, including stringently resource-constrained systems like embedded systems, smartphones, etc. An increasingly popular approach to addressing these bottlenecks is to augment the memory with processing units, often referred to as *in-memory computing/processing* or *processing in memory (PiM)*, to mitigate the cost of data transfers between the processor and the memory. PiM can enable massive parallelism via memory devices designed with multiple processing units across memory arrays [1].

Due to massive parallelism and the reduction in data movement, in-memory computing can be used in several application domains like scientific computing, healthcare, machine learning, autonomous driving, etc. Most prior research on in-memory computing focus on using the memory as an accelerator, wherein kernels are fully executed on the in-memory processing units. However, in many real-world applications, the processor must also be used, along with in-memory computing, to effectively complete the required execution [2]. This may lead to additional data movement overhead across the cache hierarchy [3]. Furthermore, in-memory computing may have limited applicability when address translation is required, as address spaces of the main memories may exceed the reach of the translation lookaside buffer (TLB), resulting in frequent and expensive page walks [4].

To mitigate these issues, prior work has proposed leveraging *compute caches* [3] or *processing in cache (PiC)*. PiC involves an architecture in which multiple word-lines in an SRAM cache are simultaneously activated to sense the resulting voltage and perform computations on the data. However, given that SRAM caches typically have high power and area requirements along with the additional overhead of augmenting the cache with processing units, SRAM is a less viable option for PiC implementations, especially in resource-constrained systems. Additionally, to avoid data corruption in SRAM during bit-line computing, the word-line voltage must be lowered, leading to an increase in the cache delay [3], [5].

Spin-Transfer Torque RAM (STT-RAM) is an emerging non-volatile memory (NVM) alternative to SRAM for implementing caches due to its lower area (40% - 80% less area) and near-zero leakage power [6]. STT-RAM has shown commercial viability and better endurance than other NVM alternatives (such as ReRAMs or SOT-RAM), making it one of the leading contenders for replacing SRAM. Furthermore, STT-RAM inherently has a higher write current than read current, preventing data corruption during bit-line computing [7]. As such, STT-RAM remains one of the more realistic technologies to implement caches in emerging computer systems. However, STT-RAM requires high write latency and write energy due to its non-volatility (i.e., the long retention time). To mitigate this overhead, the retention time can be substantially relaxed (to $< 1s$) to only satisfy the retention needs of the execution workloads' cache blocks [6], [8].

In this paper, we study PiC using relaxed retention STT-RAM caches. Unlike prior STT-RAM-based in-memory computing research, which has only considered non-volatile STT-RAM PiM implementations [7], [9], our work considers the tradeoffs of computing across the memory hierarchy, including relaxed retention STT-RAM caches and non-volatile STT-RAM main memories. Relaxed retention STT-RAMs intro-

duce new considerations (e.g., the impact of cache block lifetimes) that must be taken into account to maximize the energy benefits of STT-RAM caches. In our study, we explore different types of workloads to reveal the implications of various workload characteristics on computation efficiency. These characteristics include the CPU-dependence (due to sequential executions and complex computations), temporal locality/data reuse, and write intensity, all of which have important implications for the effectiveness of PiC vs. PiM. We show the kinds of workloads best suited for PiC/PiM, propose solutions to potential PiC design issues, and analyze the latency and energy benefits of the proposed solutions in comparison to the state-of-the-art.

In summary, we make the following key contributions:

- For the first time, we study STT-RAM-based PiC with relaxed retention at different cache hierarchy levels (specifically, L1 and L2).
- We explore different types of workloads to analyze the tradeoffs of computing at different memory hierarchy levels, including the cache hierarchy and main memory, using STT-RAM. Our analysis is performed in the context of different workload characteristics that have large implications for the effectiveness of PiC/PiM.
- We compare the energy and latency benefits of PiC using STT-RAM vs. SRAM. Experimental results reveal that STT-RAM achieves significant area savings (up to 79.86% compared to SRAM), energy savings—an average of 4.18x compared to CPU and 6% (up to 54.5%) compared to SRAM—with similar latency as SRAM-based PiC (3x improvement over CPU).
- To improve the execution of CPU-dependent workloads, for which PiC/PiM is most limited, we propose a simple optimization called *operation chaining* that enables better concurrency of execution between CPU and PiC/PiM units. The proposed approach achieves 10.19% and 7.83% average latency and energy savings, respectively, compared to the state-of-the-art.

## II. Background and Related Work

STT-RAM's cell structure consists of a transistor and a magnetic tunnel junction (MTJ). The MTJ consists of an oxide layer between two ferromagnetic layers—the free layer and the hard (or fixed) layer. The magnetization of the free layer can be changed by passing a current through it, but the magnetization of the hard layer remains fixed. The direction of the magnetization between the ferromagnetic layers determines the bit stored in the STT-RAM cell: '0' (in the parallel state or '1' in the anti-parallel state). Additional details on the characteristics and workings of STT-RAM cells can be found in prior work [10]. STT-RAMs have high area density and very low leakage power but also require high write latency and energy. This section briefly summarizes the prior work on relaxed retention STT-RAM—a candidate solution for mitigating the write latency/energy overheads in cache implementations—and processing in cache/memory (PiC/PiM).

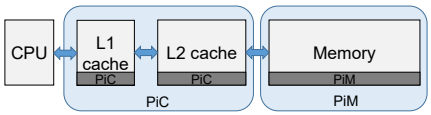


Fig. 1: The system model featuring processing in cache (PiC) implemented in the L1 and L2 caches and processing in memory (PiM) implemented in the main memory.

### A. Relaxed-retention STT-RAM caches

Prior work [8] found that relaxing STT-RAM's retention time can significantly reduce the write latency and energy. The retention time can be relaxed by changing the thickness of the free layer, magnetization saturation, or the effective anisotropy field [6], [8]. In this study, we use a technique similar to [6] to model relaxed retention STT-RAM caches.

Kuan et al. [11] found that relaxed retention STT-RAM caches can be further optimized for energy efficiency by closely matching the retention time configuration to workloads' runtime execution requirements. The authors proposed a logically adaptable retention STT-RAM (LARS) L1 cache featuring multiple retention time units and used a sampling-based algorithm to dynamically determine applications' best retention times.

### B. STT-RAM-based processing in cache/memory (PiC/PiM)

Many prior works have studied PiM using non-volatile memories (NVM) [7], [12], [13]. For example, Fan et al. [13] proposed an in-memory AES accelerator using spintronic devices. They found that, compared to CMOS circuits, spintronic devices achieve 58.6% less energy consumption. Li et al. [12] also explored bitwise PiM using NVMs like phase-change memory (PCM) and resistive RAM (ReRAM) for data-intensive applications. Jain et al. [7] studied reliable PiM under process variations using STT-RAM. Their work, which we leverage for our PiM implementation, features an STT-RAM PiM design that performs logical and addition operations.

While PiC has received much less attention than PiM, prior work [3] has shown that PiC can mitigate the data transfer overheads that might be present in PiM for some workload types. For instance, Aga et al. [3] proposed computation units within SRAM caches to mitigate data transfer overheads, improving performance and reducing energy by 1.9x and 2.4x, respectively, compared to CPU-only computing. Eckert et al. [14] proposed a Neural Cache that transforms SRAM caches into parallel compute units for deep neural network inference. Nag et al. [15] proposed GenCache, which uses SRAM-based in-cache computing to accelerate the genetic sequence alignment task. Our work is the first to explore PiC using STT-RAM and analyze the tradeoffs of STT-RAM-based computing at different levels of the memory hierarchy.

## III. Relaxed Retention Processing in Cache (PiC)

Figure 1 illustrates the system model considered in our work. We target a system wherein the computing can be performed as close to the data as possible. As such, the relaxed

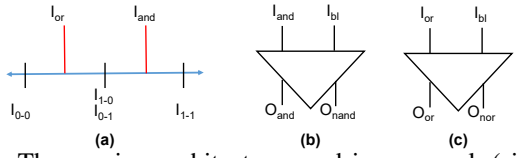



Fig. 2: The sensing architecture used in our work (similar to prior work [7]), which works for both relaxed retention and non-volatile STT-RAM computing. (a) shows the sensed current for multiple word-lines and the reference signal position; (b) and (c) show the logical compute circuits.

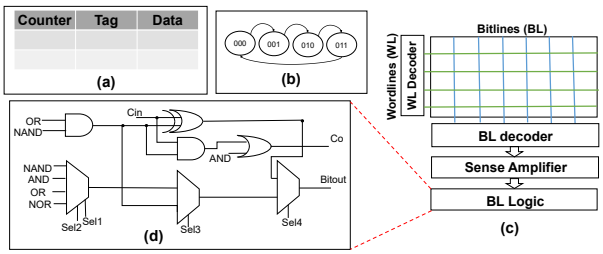



Fig. 3: (a) shows the high-level structure of a cache block (b) illustrates the cache block monitor counter implemented using a finite state machine; (c) shows the subarray of a cache with computational logic block after the sense amplifier; and (d) shows the computational logic.

retention STT-RAM L1 and L2 caches are augmented with computing circuits to enable *processing in cache (PiC)*, while the non-volatile STT-RAM main memory is augmented to enable *processing in memory (PiM)*. This paper focuses on the tradeoffs of computing at different levels of the memory hierarchy, considering the data movement overheads. This section describes our PiC architecture, retention time selection, optimization for CPU-dependent workloads, and design choices for mitigating process variations.

### A. Architecture

STT-RAM offers important advantages over SRAM with specific implications for PiC. For instance, since STT-RAMs require higher currents for writes than reads, there is a reduced chance of accidental writes when multiple word-lines are read for bit-line computing. As a result, unlike SRAM, multiple word-lines can be activated easily in STT-RAM without corrupting the data, and this helps mitigate some of the additional complexity concomitant to SRAM computing. For example, the cache operating frequency need not be reduced as would be necessary for SRAM [5].

An STT-RAM cell can be represented as a resistor based on the direction of the magnetization layers. Bits '0' or '1', stored in the STT-RAM cell, can be represented as resistors with different resistance values $R_P$ and $R_{AP}$, respectively. The difference in $R_P$ and $R_{AP}$ between the stored bits is called the *tunnel magneto-resistance (TMR) ratio*. While sensing, a bit-line will have different current values based on the stored bit. Similarly, sensing the currents for multiple word-lines will result in different output current values, as shown in Figure 2a. Based on bits 1 or 0 in these cells, there are three possible current values: $I_{0\text{-}0}$, $I_{1\text{-}0}$, or $I_{0\text{-}1}$ and $I_{1\text{-}1}$. As such, reference currents can be used to directly compute AND/NAND operations (Figure 2b) or NOR/OR operations (Figure 2c).

**PiC architecture.** Figure 3 presents a high-level overview of our relaxed retention PiC architecture. As seen in Figure 3a, each cache block has a retention time counter to prevent data corruption due to data expiry. We incorporated this counter to prevent data from becoming unstable if the retention time elapses before the data is evicted. The counter is implemented as an $N$-state finite state machine (Figure 3b) that begins at the initial state when the cache block is written, up-counts until the retention time is about to expire, and raises a flag to evict the block or write back to a lower memory level if dirty. We assume $N$=4 in our study, resulting in a 2-bit per block overhead (Figure 3b), with the counter's clock period set as 18.75$\mu$s [11], [16]. The monitor counter only incurs a per-block area overhead of 0.78%. Designers can choose to increase the counter size $N$ to allow for greater precision control through the same clock input. Increasing the counter size does not significantly impact the cache's area, routing costs, or critical path.

The cache is divided into mats and further into subarrays. The subarrays in each mat consist of one sense amplifier for a group of word-lines, and each word-line has one stored array to perform parallel computations. The architecture activates two word-lines simultaneously to perform arithmetic or logical operations using the combinational logic shown in Figure 3d. The architecture supports addition and logical operations, while complex operations like multiplication are performed on the CPU for a simpler PiC implementation. Since the operations implemented are associative, they can be performed by simply sensing multiple word-lines, and the output of the desired operation can be selected from multiplexers orchestrated by the cache controller.

**Supporting regular cache operations.** Given a PiC-enabled system, it may still be necessary to perform regular cache operations for CPU-based computing. To this end, we present two design choices. The first involves using additional sense amplifiers to sense bit '0' or '1' along with the sense amplifiers described in Figure 2. This additional sense amp will increase the area overhead but lead to lower access latency since the compute elements of the PiC operations can be bypassed. Alternatively, the PiC sense amplifiers ((Figure 2c)), which have a reference current tuned to read two cache word-lines, can be used—one word-line is the one being read in the cache operation, and the second is set as bit '0'. The sensed output from the sense amplifier is obtained like an *OR* output in the PiC-based computation (i.e., compared to the reference $I_{OR}$ and read through the multiplexer's OR output (Figure 2d)) . While this option might reduce the area overheads, it might slow down the cache read operations if the compute elements have a high latency overhead. For our work, we used the latter design choice, since the compute elements used in this design are simple and do not impose enough latency overhead to impact the cache access cycles. The write operations for CPU-based computing and PiC use the same design circuits since

only one word-line is activated during cache block writes or while storing the PiC computation results.

**Scaling the parallel computations.** To increase the number of parallel units, the number of subarrays and the cache geometry must be modified. This is relatively straightforward for SRAM caches, for which the number of subarrays and sense amplifiers can easily be increased [3]. However, relaxed retention STT-RAMs must also take into account the cache block monitor counters, which reset when a write operation occurs. In PiC, the counter is reset whenever the computed results are stored in the cache block. If a cache block is partially updated, the unchanged words in the cache block may expire without resetting the cache block monitor counter. In such cases, a complete cache block must be used to store the computed results to avoid discrepancies in block monitor counter updates and thus avoid data corruption due to an elapsed retention time. Therefore, the number of compute units in relaxed retention STT-RAM PiC must be a multiple of the cache block size. For example, consider a 64B block cache that can perform 16 parallel 32-bit integer computations. To increase the number of parallel computations to 32, the cache geometry can be modified to increase the number of subarrays such that two cache blocks (128B) are updated simultaneously. As such, the counter remains simple and seamlessly integrated with relaxed retention caches for both CPU and PiC operations.

### *B. Determining the best retention time*

In a relaxed retention STT-RAM, it is imperative that the retention time be sufficient for the cache block lifetimes of the executing workloads. A longer retention time than necessary will incur write overheads, while a shorter retention time will result in premature expiry of data blocks, leading to high miss rates and data movement overheads. Prior works mapped the retention time to workloads' execution characteristics in traditional CPU-based processing [11]. However, we empirically found that such a scheme is not needed for PiC. In traditional computing, the retention time depends on an application's average cache block lifetime and how frequently the data blocks are accessed. For PiC, the cache begins processing as soon as the data is made available in the cache. Thus, the retention time requirement for PiC depends on the time taken to bring the required operands into the cache for computation. For instance, given a computation $c = a + b$, the block containing word $a$ (say, $blockA$) only needs to remain in the cache long enough to bring $blockB$ (containing word $b$) into the cache to complete the computation. If it takes 100 cycles (or 50ns at 2GHz) to bring $blockB$ from memory to cache, then for a 32kB cache size, in the worst-case scenario, a 25.6$\mu$s retention time is required.

To determine the appropriate retention times, we first analyzed the workloads to determine the average miss latency for each cache hierarchy level. Based on this miss latency, we selected the retention time such that data block expiration does not occur during PiC computations. Based on our analysis, we found that 75$\mu$s sufficed for the L1 cache and 10ms sufficed for the L2 cache in both PiC and CPU-based computing while minimizing the premature expiry of data blocks. Note that different retention times may be required for different sets of workloads. For example, workloads with high data reuse may require data to be fetched more frequently from the cache rather than from the main memory. In such cases, a shorter retention time might suffice for PiC. But the choice of specific retention times is orthogonal to the rest of our analysis.

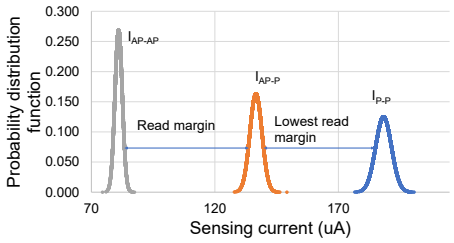



Fig. 4: The probability distribution function of the sensing current for PiC STT-RAM under 5% process variation for 10000 samples.

### *C. Mitigating the effects of process variation*

When performing a read operation, bits 0 and 1 have different resistance values (TMR ratio), which change the sense amplifier's output current. We modeled multiple retention times by varying the STT-RAM cell parameters like the free layer thickness and anisotropy constant, $Hk$, while keeping the TMR ratio constant. This enabled us to use the same sense amplifier design for all the retention times. For PiC computations, the sense amplifier must sense three levels, $I_{0\text{-}0}$, $I_{1\text{-}0}$, or $I_{0\text{-}1}$ and $I_{1\text{-}1}$, as described in Section III-A. Previous work [7] found that a TMR ratio of 124% was sufficient for bit-line computing in STT-RAM to perform reliable read operations for multiple word-lines. However, we used a TMR ratio of 150%, which we found to further increase the difference in the sensed current output for bits 0 and 1, thereby enabling more reliable and distinct current levels.

To study the impact of process variations, we performed Monte-Carlo simulations [17] for 10000 samples of varying STT-RAM cell resistance values using SPICE. Figure 4 presents our results on the difference in current levels for STT-RAM PiC under the process variations. For our experiments, we set the $R_{AP}$ and $R_P$ under 5% process variation similar to prior work [7]. As seen in the figure, $I_{AP\text{-}AP}$ passes through the highest resistance $R_{AP\text{-}AP}$ resulting in a low sensing current. $I_{AP\text{-}AP}$ has a low standard deviation in sensing current under process variation, resulting in a higher probability distribution. Similarly, $I_{P\text{-}P}$ passes through the lowest resistance $R_{P\text{-}P}$ and has a high standard deviation in the sensing current, resulting in the lowest probability distribution function. We also observed that $I_{AP\text{-}AP}$, $I_{AP\text{-}P}$ or $I_{P\text{-}AP}$ and $I_{P\text{-}P}$ are significantly distinct even under process variations. The read margin between $I_{AP\text{-}P}$ and $I_{P\text{-}P}$ is smaller than the read margin between $I_{AP\text{-}P}$ and $I_{AP\text{-}AP}$. We used the smallest read margin to tune the TMR ratio to increase the reliability of the sensing operation.

However, a high TMR ratio can increase the switching energy used to flip the bit of the STT-RAM cell. Prior works used relaxed retention time to reduce the switching energy

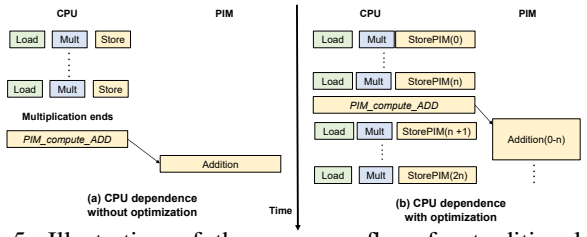



Fig. 5: Illustration of the program flow for traditional vs. operation-chained PiC/PiM.

(write energy) of STT-RAM. When the retention time is relaxed, the switching energy can either be set by reducing the write current or the write pulse [6], [18]. Prior studies have found that reducing the write pulse width while keeping the switching current constant helps mitigate the read/write errors in relaxed retention STT-RAM [6]. We used a similar approach to relax the retention time for PiC by keeping a constant switching current for all the retention times and only varying the switching pulse of the STT-RAM cell. As such, the switching current is set much higher than the read current to mitigate bit-flipping errors when reading multiple word-lines.

### D. Operation chaining for CPU-dependent workloads

Given the potential overhead of CPU-dependent workloads, we sought to improve the parallelism to minimize the overhead of waiting for results from the CPU. As seen in prior work [2], traditional PiM divides the program into PiM and CPU execution portions; the PiM begins only once the CPU execution is complete. This computation model is depicted in Figure 5. By analyzing the workloads, we found that the CPU-dependent workloads could be optimized to better exploit parallel processing between the PiC/PiM and CPU execution units. The optimization, called *operation chaining*, is inspired by *vector chaining* architectures [19]. As illustrated in Figure 5, operation chaining allows interim CPU results to be used for PiC/PiM computations without introducing additional memory references. After each computation, the processor stores the intermediate results in the memory or cache for the PiC/PiM computations. The CPU communicates via a PiC/PiM controller using a *Compute* signal to start the PiC/PiM computations and receives a *DONE* signal which indicates that the computations have been successfully completed. With this, the execution latency of PiC/PiM is effectively hidden behind the processor's execution latency.

Operation chaining is enabled by the compiler, which detects the data dependencies and ensures the absence of potential hazards. The programs are augmented with new instructions: *StorePIM* (to store data in the appropriate PiC/PiM location) and *Compute_Inst_PIM* (to initiate execution in the PiC/PiM architecture).

## IV. Experiments

### A. Workloads

To model the behavior of real-world applications, we used eight workloads with different characteristics. Table I depicts workloads and their input sizes. For our analysis, we classify the workloads into three groups. *CPU-dependent workloads* are those that have large portions with characteristics that make them more efficiently executed by the CPU. These characteristics include low instruction-level parallelism, pointer-chasing operations, complex branch conditions, or complex computations for which the in-memory computing design is ill-equipped (e.g., multiply operations in our work). *CPU-independent* workloads are simple kernels that can run entirely via PiC/PiM. Prior research on in-memory computing is dominated by these types of workloads [3], [9]. We further grouped the CPU-independent workloads into high data reuse and low data reuse, enabling us to evaluate the tradeoffs involving the data movement overheads for relaxed retention STT-RAM PiC and non-volatile STT-RAM PiM. The workloads selected are commonly used in applications like image/signal processing, data querying, etc.

TABLE I: Workloads used in our experiments

| Category | Kernel | Input size |
|---|---|---|
| CPU-dependent | Kmeans nearest neighbor (*KNN*) | $10^5$ nodes |
| | 2D convolution (*conv*) | $10^6$ samples |
| | Histogram (*hist*) | $10^6$ samples |
| | Root-mean square error (*rmse*) | $10^6$ samples |
| CPU-independent, high data reuse | Binarized neural network (*bnn*) | $10^6$ samples |
| | Matrix addition (*mat_add*) | $10^6$ samples |
| | String Comparison(*string*) | 2,409,780 letters |
| CPU-independent, low data reuse | Carryless multiplication (*cmul*) | $10^6$ samples |

### B. Experimental methodology

To model the PiC/PiM computation logic, we used SPICE simulations with 22nm CMOS libraries. We used NVSim [20] to obtain the STT-RAM/SRAM cache and the STT-RAM memory access latency and energy. We modified gem5 [21] to implement relaxed retention STT-RAM caches and to model SRAM caches. The gem5 statistics were then integrated with McPAT [22] to obtain the total system power. We modeled a processor like ARM Cortex A72 with a 2GHz clock and 8GB memory, of which 512MB is used for PiM.

Table II shows the cache and memory configurations, read, write, and computation latencies for each operation at each level of the memory hierarchy. The logical operations take one cycle since they require approximately 120ps to finish the bitwise logical operations. To save the area and energy for ADD operations, we use the Ripple Carry Adder design, which computes one bit at a time and then sends the carry bit to higher significant bits. STT-RAM requires fewer cycles than SRAM for ADD operations because of a longer slack within each STT-RAM read cycle, enabling more bit operations per cycle. The energy numbers are calculated as the energy needed to access the subarrays and then the bit-line and cell of the memory device. Logical and ADD computation energies include the energy required to access the cache, perform the required computations, and store the results in a subarray. Like prior work [3], we assume that the processor's execution unit is powered off to conserve power during PiC/PiM.

## V. Results and Analysis

In this section, we first compare operation chaining with prior PiC/PiM computing and then analyze the tradeoffs of

TABLE II: Cache and memory configurations

| Memory hierarchy | L1 cache 32kB-64B-4 | | L2 cache 1MB-64B-8 | | | Memory 512MB size |
|---|---|---|---|---|---|---|
| Memory Device | SRAM | STT-RAM | SRAM | STT-RAM | | STT-RAM |
| Retention time | – | 75$\mu$s | – | 75$\mu$s | 10ms | 5years |
| Read latency (cycles) | 1 | 1 | 2 | 2 | 2 | 32 |
| Write latency (cycles) | 1 | 2 | 2 | 3 | 4 | 56 |
| Logical operation (cycles) | 3 | 3 | 4 | 5 | 6 | 88 |
| Add operation (cycles) | 18 | 15 | 19 | 15 | 16 | 97 |
| Read energy per bit (in pJ) | 0.125 | 0.086 | 1.77 | 0.75 | 0.75 | 24.55 |
| Write energy per bit (in pJ) | 0.19 | 4.69 | 0.62 | 9.647 | 15.604 | 640.89 |
| Logical computation energy per bit (in pJ) | 0.915 | 5.376 | 2.997 | 10.997 | 16.954 | 666.045 |
| Add computation energy per bit (in pJ) | 1.355 | 5.816 | 3.437 | 11.437 | 17.394 | 666.49 |
| Leakage power (mW) | 43.95 | 17.63 | 1168.95 | 182.8 | 182.2 | 222.36 |

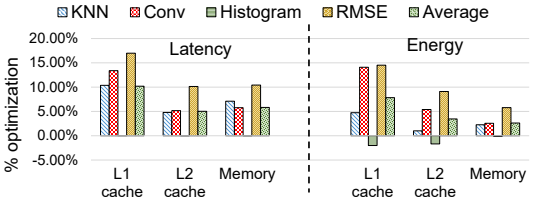



Fig. 6: Latency and energy savings of CPU-dependent workloads using operation chaining for PiC/PiM at various memory hierarchy compared to prior work (no operation chaining).

PiC using STT-RAM vs. SRAM. Thereafter, we compare STT-RAM-based PiC with PiM and describe the overheads. Finally, we present the PiC and PiM overheads to understand the tradeoffs of the two approaches.

### *A. Comparison between operation-chained and traditional PiC/PiM (prior work)*

First, we compare the optimized method for CPU-dependent workloads, using operation chaining (Figure 5), with the traditional PiM as done in prior work [2]. The comparison is performed for different memory hierarchy levels, using STT-RAMs for the cache hierarchy (L1 and L2) and memory. We assume that the processor can run simultaneously along with PiC/PiM elements to fully exploit the benefits of operation chaining. Although PiC has only previously been implemented in SRAM [3], [14], we used an STT-RAM PiC implementation to analyze the benefits of operation chaining. Unlike traditional PiC/PiM computing, the processor's execution unit remains powered on while using operation chaining.

Figure 6 presents the latency and energy savings of the operation-chained PiC/PiM for L1 cache, L2 cache, and memory, compared to the traditional PiC/PiM. On average, across the CPU-dependent workloads, operation chaining improved the overall latency for computing in L1 cache, L2 cache, and memory by 10.19%, 5.02%, and 5.82%, respectively. The traditional PiC involved a lot of data written back to lower memory levels that had to be reloaded into the cache, while PiC with operation chaining reduced data movement (by up to 10.21%) and increased compute unit utilization. The highest latency improvements were observed for $RMSE$ (16.98%, 10.13%, and 10.43% for L1, L2, and memory, respectively), which required multiple data transfers to the processor for more complex computations (e.g., square and square root operations). In the worst case, operation chaining did not offer any savings for $histogram$ because most of the application was run on the CPU due to complex or sequential operations.

Operation chaining reduced the energy by 7.83%, 3.45%, and 2.62%, respectively for L1, L2 cache, and memory, compared to traditional PiC/PiM. Like the latency, the highest energy improvement was observed for $RMSE$, while operation chaining slightly degraded the energy for $histogram$ by up to 2.02%. The energy increased because the CPU and PiC/PiM execution units were simultaneously active and the latency savings did not offset the extra power incurred by the PiC/PiM execution units. Given the overall superiority of the operation-chained PiC/PiM over traditional PiC/PiM, in what follows, we perform an analysis of computing across the memory hierarchy using the operation-chained PiC/PiM.

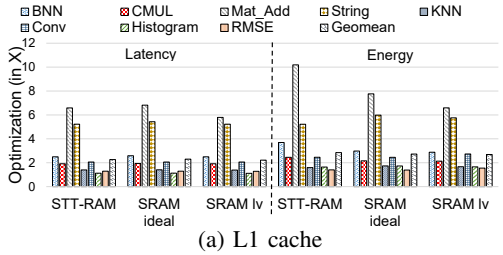
### *B. Comparison of different PiC candidates*

In this subsection, we explore candidates for PiC, including relaxed retention STT-RAM, an SRAM design used in prior work with a low-voltage word-line (SRAM$_{lv}$) [5], and a theoretical ideal SRAM cache model (SRAM$_{ideal}$) that achieves the least data corruption during bit-line computing. SRAM$_{lv}$ has 50% more delay than SRAM$_{ideal}$, and requires 20% lower dynamic energy for its operations. Although the 10ms retention was sufficient for the L2 cache (Section III-B), we also explored a 75$\mu$s L2 cache for additional analysis.

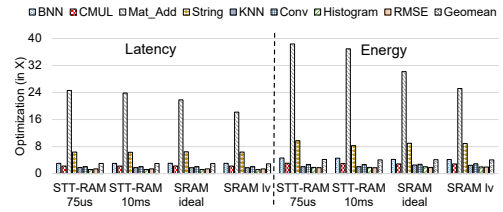
Figure 7a depicts the optimization achieved using different L1 PiC implementations compared to CPU-only computing for all the workloads considered. On average, STT-RAM, SRAM$_{ideal}$ and SRAM$_{lv}$ reduced the latency by 2.27x, 2.3x, and 2.22x, compared to the CPU. SRAM$_{ideal}$ had the best execution time due to low write overheads compared to STT-RAM. Compared to STT-RAMs, SRAM$_{lv}$ slightly increased the latency by 1.95% due to longer delays.

STT-RAM PiC performed much better with respect to energy. STT-RAM, SRAM$_{ideal}$, and SRAM$_{lv}$ reduced the energy by 2.84x, 2.72x, and 2.68x, respectively, compared to CPU. STT-RAM outperformed SRAM$_{ideal}$ by 4.56%. Although SRAM$_{lv}$ had lower dynamic power than SRAM$_{ideal}$, it had a higher total energy because it ran slower. We observed the highest improvement for matrix addition ($Mat\_add$), which exhibited high data reuse—and more computations per data movement—during sum accumulation. The L1 cache reduced the latency for $Mat\_add$ by 6.58x, 6.819x, and 5.79x, and reduced the energy by 10.19x, 7.76x, and 6.59x using STT-RAM, SRAM$_{ideal}$, and SRAM$_{lv}$, respectively. However, CPU-dependent workloads had the lowest optimizations due to high amounts of data movement (55% - 75% of total execution) and smaller portions of the workloads running on PiC/PiM. The latency improved for the CPU-dependent workloads by an average of 1.43x, 1.44x, 1.42x, and energy was reduced by 1.73x, 1.78x, and 1.84x using STT-RAM, SRAM $_{ideal}$, and SRAM$_{lv}$ respectively.

We observed higher optimization in L2 PiC than in L1 PiC, due to more parallel units and lower data movement overheads




(a) L1 cache




(b) L2 cache

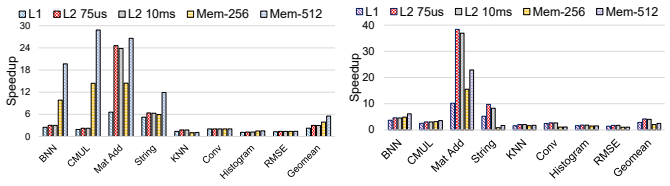
Fig. 7: L1/L2 STT-RAM, ideal SRAM (SRAM$_{ideal}$) and low voltage word-line SRAM (SRAM$_{lv}$) compared to CPU.

from memory. As seen in Figure 7b, STT-RAM$_{75\mu s}$, STT-RAM$_{10ms}$, SRAM$_{ideal}$, and SRAM$_{lv}$ reduced the latency by 3x, 2.98x, 2.959x, and 2.889x, respectively, compared to CPU. STT-RAMs achieved higher speedup than SRAM for ADD operations, due to faster read operations that enabled longer slack to perform more bit additions per cycle. SRAM$_{ideal}$ achieved the fastest speedup for logical instructions as seen in $BNN$, $CMUL$, and $String$ applications due to SRAM's faster write operations. For these applications, STT-RAM$_{75\mu s}$ and STT-RAM$_{10ms}$ only degraded performance by 0.73% and 1.44%, respectively. SRAM$_{low}$ achieved low speedup for both logical and ADD instructions. Compared to SRAM$_{lv}$, STT-RAM$_{75\mu s}$ and STT-RAM$_{10ms}$ reduced the latency by 35.23% and up to 31.06% for $Mat\_add$ application.

The energy savings from L2 PiC were similarly quite substantial, as seen in Figure 7b. The energy savings compared to CPU-only computing for STT-RAM$_{75\mu s}$, STT-RAM$_{10ms}$, SRAM$_{ideal}$, and SRAM$_{lv}$ were 4.19x, 4.05x, 4.12x, and 4.05x, respectively. The energy savings were highest for $Mat\_add$ 38.38x, 36.94x, 30.19x, and 25.18x with STT-RAM$_{75\mu s}$, STT-RAM$_{10ms}$, SRAM$_{ideal}$, and SRAM$_{lv}$, respectively. SRAM$_{ideal}$ outperformed STT-RAM$_{10ms}$ because of the high write energy overheads incurred by STT-RAM's higher retention time and SRAM$_{ideal}$'s lower static energy overhead due to faster program execution. However, STT-RAM$_{10ms}$ performed marginally better than SRAM$_{low}$ by an average of 0.5% indicating that the static energy overhead of slower SRAM$_{low}$ is higher than the dynamic energy overhead of STT-RAMs. These results reveal the superiority of STT-RAM for PiC and also indicate that a shorter retention time in the L2 cache suffices for PiC, unlike CPU-based computing, which requires longer retention times in the L2 cache [6], [23].

### C. PiC vs. PiM

We explore the best level of the hierarchy for STT-RAM-based PiC/PiM. For these experiments, we compare PiC with 75$\mu$s retention time for L1/L2 cache, 10ms for L2 cache, and




(a) Latency (b) Energy

Fig. 8: Relaxed retention STT-RAM PiC and non-volatile STT-RAM PiM compared to CPU. Memory that does 256 and 512 integer computations is called *Mem-256* and *Mem-512*.

non-volatile PiM. The L1 cache supports 16 32-bit computations, L2 supports 64 32-bit computations, and we explored memory with the capability for 256 or 512 32-bit computations (called *Mem-256* and *Mem-512*, respectively).

Figure 8 presents the latency and energy results for computing across the memory hierarchy. As seen in Figure 8a, L1 cache, L2$_{75\mu s}$, L2$_{10ms}$, Mem-256, and Mem-512 improved the latency by 2.26x, 3x, 2.98x, 3.90x, and 5.53x, respectively, compared to CPU-only computing. A closer look at the workloads showed that different parts of the hierarchy were preferred for different types of workloads. The average improvement for CPU-dependent workloads was 1.46x, 1.61x, 1.61x, 1.49x, and 1.52x for L1 cache, L2$_{75\mu s}$, L2$_{10ms}$, Mem-256, and Mem-512, respectively. L2 PiC achieved the highest optimization for CPU-dependent workloads due to the reduction in data movement overhead from the processor and a large number of parallel units. However, for CPU-independent workloads, PiM had the fastest execution due to low data movement and a large number of parallel units. For CPU-independent workloads, we observed a speedup of 4.05x, 9.06x, 8.85x, 11.17x, and 21.76x, respectively. This shows that Pic works best for CPU-dependent workloads, whereas PiM works best for CPU-independent workloads despite having high access latency costs, as seen in Table II.

Contrary to latency, the L2 PiC outperformed the PiM in energy savings, as seen in Figure 8b. On average, L1 cache, L2$_{75\mu s}$, L2$_{10ms}$, Mem-256, and Mem-512 reduced the energy by 2.84x, 4.18x, 4.05x, 2.04, and 2.48x, respectively, compared to CPU. We also observed that even though PiM's and PiC's execution latencies could be hidden behind operation chaining, the number of computations that are performed remained the same. This puts PiM at a disadvantage due to very high write energy overheads. Furthermore, high data reuse workloads also require additional writes within the same memory hierarchy level, which greatly increases the dynamic energy for the non-volatile memory and favors PiC by increasing the computations per data transferred. PiM only achieved better energy savings for two applications, $BNN$ and $CMUL$, both of which mainly perform logical operations and feature few write operations.

### D. Overhead

The implemented PiC/PiM compute units had a critical path of 120ps and did not increase any of the cache access latencies. The energy per bit for logical and ADD operations

was 0.6pJ and 1.04pJ, respectively. For STT-RAM L1, STT-RAM L2, and STT-RAM memory, the compute elements were only 11.16%, 3.54%, and 0.55% of the total energy. For a 6T SRAM subarray of size 512x512, the compute unit took 3.7% of the area. For STT-RAMs, which are twice as dense as SRAM, our compute units required 6.37% of the total subarray area. The STT-RAM cache consumed 43.47% and 79.86% less area than SRAM for L1 and L2 cache, respectively. These area savings by STT-RAMs give greater flexibility to incorporate more complex computational units in resource-constrained systems. Furthermore, SRAM-based PiC incurs additional performance overheads to reduce data corruption issues [5]; these overheads can also slow down CPU-based computing. Overall, STT-RAMs represent a more robust and low-overhead solution for PiC implementations.

## VI. Conclusion

In this paper, we performed the first study of STT-RAM cache as a candidate for processing in cache (PiC). We compared relaxed retention STT-RAM PiC to SRAM and compared STT-RAM PiC with non-volatile STT-RAM processing in memory (PiM) to analyze the tradeoffs of computing at different memory hierarchy levels. For our analysis, we explored three types of workload: CPU-dependent, CPU-independent with low data reuse, and CPU-independent with high data reuse. Our analysis reveals that STT-RAM offers an excellent opportunity for energy- and area-efficient PiC while providing latency benefits similar to those of SRAM. We also found that the choice of PiC/PiM is impacted by the executing workloads' characteristics. For instance, STT-RAM PiC outperforms PiM for latency optimization in CPU-dependent workloads with low ILP. This study shows that STT-RAM-based PiC offers much promise and warrants additional studies for effective implementation in emerging resource-constrained systems.

Future work involves studying the impact of workloads in which the PiC/PiM bit-line data are not aligned or in the same subarray. We also plan to explore heterogeneous retention cache architectures for PiC and develop scheduling algorithms for multi-application hierarchical PiC/PiM.

## Acknowledgment

This work was partly supported by the National Science Foundation (NSF) under grant CNS-1844952. Any views expressed in this material are those of the authors and not necessarily of the NSF.

## References

[1] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[2] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan *et al.*, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 316–331.

[3] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 481–492.

[4] J. Picorel, D. Jevdjic, and B. Falsafi, "Near-memory address translation," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. Ieee, 2017, pp. 303–317.

[5] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, "A 28 nm configurable memory (tcam/bcam/sram) using push-rule 6t bit cell enabling logic-in-memory," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, 2016.

[6] Z. Sun, X. Bi, H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu, and W. Wu, "Multi retention level stt-ram cache designs with a dynamic refresh scheme," in *proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture*, 2011, pp. 329–338.

[7] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, "Computing in memory with spin-transfer torque magnetic ram," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 470–483, 2017.

[8] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient stt-ram caches," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, 2011, pp. 50–61.

[9] F. Parveen, Z. He, S. Angizi, and D. Fan, "Hielm: Highly flexible in-memory computing using stt mram," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2018, pp. 361–366.

[10] K. C. Chun, H. Zhao, J. D. Harms, T.-H. Kim, J.-P. Wang, and C. H. Kim, "A scaling roadmap and performance evaluation of in-plane and perpendicular mtj based stt-mrams for high-density cache memory," *IEEE journal of solid-state circuits*, vol. 48, no. 2, pp. 598–610, 2012.

[11] K. Kuan and T. Adegbija, "Energy-efficient runtime adaptable l1 stt-ram cache design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 6, pp. 1328–1339, 2019.

[12] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[13] D. Fan, S. Angizi, and Z. He, "In-memory computing with spintronic devices," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 683–688.

[14] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *2018 ACM/IEEE 45Th annual international symposium on computer architecture (ISCA)*. IEEE, 2018, pp. 383–396.

[15] A. Nag, C. Ramachandra, R. Balasubramonian, R. Stutsman, E. Giacomin, H. Kambalasubramanyam, and P.-E. Gaillardon, "Gencache: Leveraging in-cache operators for efficient sequence alignment," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 334–346.

[16] D. Gajaria and T. Adegbija, "Evaluating the performance and energy of stt-ram caches for real-world wearable workloads," *Future Generation Computer Systems*, 2022.

[17] R. L. Harrison, "Introduction to monte carlo simulation," in *AIP conference proceedings*, vol. 1204, no. 1. American Institute of Physics, 2010, pp. 17–21.

[18] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache revive: Architecting volatile stt-ram caches for enhanced performance in cmps," in *DAC Design Automation Conference 2012*. IEEE, 2012, pp. 243–252.

[19] J. J. Dongarra, F. G. Gustavson, and A. Karp, "Implementing linear algebra algorithms for dense matrices on a vector pipeline machine," *Siam Review*, vol. 26, no. 1, pp. 91–112, 1984.

[20] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[21] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[22] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd annual ieee/acm international symposium on microarchitecture*, 2009, pp. 469–480.

[23] K. Kuan and T. Adegbija, "Halls: An energy-efficient highly adaptable last level stt-ram cache for multicore systems," *IEEE Transactions on Computers*, vol. 68, no. 11, pp. 1623–1634, 2019.