# Thermal-Aware Design and Management for Search-based In-Memory Acceleration

Minxuan Zhou, Mohsen Imani, Saransh Gupta, and Tajana Rosing
CSE Department, UC San Diego, La Jolla, CA 92093, USA
{miz087, moimani, sgupta, tajana}@ucsd.edu

## ABSTRACT

Recently, Processing-In-Memory (PIM) techniques exploiting resistive RAM (ReRAM) have been used to accelerate various big data applications. ReRAM-based in-memory search is a powerful operation which efficiently finds required data in a large data set. However, such operations result in a large amount of current which may create serious thermal issues, especially in state-of-the-art 3D stacking chips. Therefore, designing PIM accelerators based on in-memory searches requires a careful consideration of temperature. In this work, we propose static and dynamic techniques to optimize the thermal behavior of PIM architectures running intensive in-memory search operations. Our experiments show the proposed design significantly reduces the peak chip temperature and dynamic management overhead. We test our proposed design in two important categories of applications which benefit from the search-based PIM acceleration - hyper-dimensional computing and database query. Validated experiments show that the proposed method can reduce the steady-state temperature by at least 15.3 °C which extends the lifetime of the ReRAM device by 57.2% on average. Furthermore, the proposed fine-grained dynamic thermal management provides 17.6% performance improvement over state-of-the-art methods.

## CCS CONCEPTS

• **Computer systems organization → Architectures**; • **Hardware →** *Emerging technologies*;

**ACM Reference Format:**
Minxuan Zhou, Mohsen Imani, Saransh Gupta, and Tajana Rosing. 2019. Thermal-Aware Design and Management for Search-based In-Memory Acceleration. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3316781.3317923

## 1 INTRODUCTION

In today's big-data era, data movements between off-chip memory and computing cores dominate energy consumption and performance of conventional computing systems [1]. Processing-in-memory (PIM) is a promising technique to address this issue by processing data locally in memory [1–3]. Most PIM technologies exploit the analog characteristic of dense and efficient non-volatile memories (NVMs) [1, 4, 5]. For example, work in [4] exploited the current-based switching of the NVM devices to implement NOR-based operations. Work in [5–8] extended this bitwise operation to support 32-bit additions and multiplications in order to accelerate big data applications. Other than normal arithmetic operations, recent PIM architectures utilize complex operations such as associative search to complete data-intensive tasks in memory. Such in-memory search operations can be used to accelerate various important applications such as machine learning and query processing [9–14]. An in-memory search operation may consume a massive amount of power because it activates all memory rows. For instance, the energy consumed by an in-memory search operation on 32 ReRAM rows (1024 bits/row) in 45nm is 2740fJ [15], which means the power consumption of such operations may reach as high as 2.74W at 1GHz frequency. Considering the large number of memory rows required for search operations in large datasets, such power consumption may cause serious thermal issues in the memory chip especially when using the state-of-the-art 3D stacking technologies [16]. Temperature plays an important role in not only reliability but also endurance of ReRAM [17]. Handling the thermal-related issues becomes critical for emerging PIM acceleration based on in-memory associative search.

Temperature optimization for PIM acceleration should come from both off-line and online stages. Off-line data allocation determines high power-consuming regions of ReRAM chip which may cause extremely various thermal behaviors while an online method guarantees the dynamic temperature can be controlled in a reasonable range. In this work, we first explore application data allocations in the PIM architecture to optimize its steady-state temperature. Due to the large design space, we propose a genetic algorithm to efficiently search for a near thermal-optimal data allocation. Nevertheless, such thermal-optimal data allocation may still fail to meet the temperature limitation, which makes dynamic thermal management (DTM) necessary to further control the run-time temperature behavior. Traditional methods to manage the chip temperature include dynamic voltage and frequency scaling (DVFS) [18], task allocation [19], and memory access control [20]. However, directly applying DVFS may significantly sacrifice the performance of the PIM architecture because other operations including normal memory operations and arithmetic PIM operations will slow down. In fact, there is no solution available for controlling the power consumption of PIM applications which are dominated by in-memory searches.

To enable an efficient DTM, we propose a new search design which divides each in-memory search operation into a configurable number of searches, each of which is applied on a subset of the table. Based on the proposed design, we implement a fine-grained management mechanism which adjusts search operations handled by different vaults based on dynamic temperatures. We test all proposed mechanisms in several emerging applications accelerated by the PIM architecture from fields of machine learning and database query. Based on our experiment, the thermal-optimal data allocation scheme found by our algorithm reduces the steady-state temperature by at least 15.3 °C extending the lifetime of the ReRAM device by 57.2% on average. Our evaluation shows that the proposed dynamic management approach can improve the PIM performance by at least 17.6% as compared to state-of-the-art DTM methods.

*DAC '19, June 2–6, 2019, Las Vegas, NV, USA*
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317923

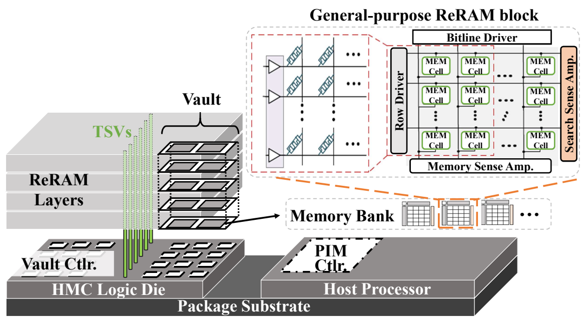

**Figure 1: The structure of HMC-like [16] 3D stack memory consisting of general purpose PIM-enable ReRAM.**

## 2 BACKGROUND

### 2.1 ReRAM-based PIM Architecture

To solve the memory wall issue, emerging memory technologies have been exploited as not only fast and scalable memory [17, 20, 21], but also as in-memory processing units. ReRAM is one of the most popular technologies which can efficiently handle these operations. In this work, we adopt a Hybrid Memory Cube (HMC)-like [16] memory design which has been used in several recent emerging memory architectures [17, 20, 21] (Figure 1). The architecture contains multiple memory layers and one logic layer where the data is transferred by the Through-Silicon-Vias (TSVs) vertically from the memory to the controller placed on the logic layer. The architecture is divided into multiple vaults, each of which contains multiple banks. We adopt a 2.5D design which places the host processor in a separate stack with a centralized PIM controller. The PIM controller communicates with vault controllers based on the address of instructions and each vault controller processes operations in the vault. Each bank consists of multiple memory blocks each of which is an array of ReRAM cells. The peripheral circuits in the ReRAM block handle data stored in ReRAM cells based on the operation. We utilize a general-purpose ReRAM block design which enables various PIM operations including bit-wise operations, additions, multiplications, and associative searches [11].

### 2.2 Search-based PIM Applications

In this paper, we focus on the emerging ReRAM-based PIM architecture to accelerate brain-inspired computing [13] and query processing [11] which requires intensive in-memory associative search operations.

**Brain-inspired computing:** Hyperdimensional (HD) computing is based on understanding the fact that brains compute with *patterns of neural activity* which are not readily associated with numbers [22]. However, due to the very large size of the brain's circuits, such neural activity patterns can only be modeled with points of high-dimensional space (e.g., $D$=10,000). The HD computing can perform the classification task using two main modules; encoding and associative search [23]. The encoding module maps input data into high-dimensional space, hypervector, then the training module combines all hypervectors in order to generate a binary hypervector representing each class. For example, an HD model with $k$ classes will end up having $k$ hypervectors where each corresponds to one of the classes. Each block implements the encoding and training using in-memory bit-wise operations (XOR) [4] and in-memory additions between the vectors in 10,000 dimensions (Figure 2a) [24]. After training, the classification can be performed using a set of search-based operations. A search block compares the similarity of an encoded data with all pre-trained class hypervectors (Figure 2b).

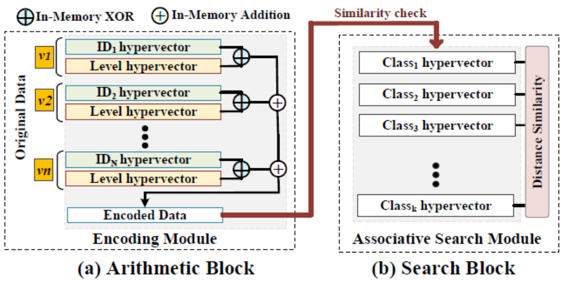

**Figure 2: The PIM architecture processing HD computing in (a) arithmetic and (b) search blocks.**

The goal of the search block is to find a class hypervector which has the highest similarity to the pattern of the encoded data.

**Query processing:** In data management systems, the execution time of queries tends to increase linearly or sometimes exponentially as more records are stored in a single server instance. Data movement is the main bottleneck of current computing systems when the size of data increases over the cache capacity of the processing core [25]. Processing in-memory architecture accelerates the query processing by supporting all essential query functionality in memory [11]. Similar to HD computing, query processing also requires both arithmetic and search blocks. A query application may involve both an exact search and a search for minimum/maximum values. The PIM architecture supports different types of search operations by exploring different sense amplifiers for a content addressable memory block [10, 13]. The results of the search operation are written into other memory blocks which are reserved to perform the computation on the selected data. The example of operations is the addition, multiplication, and bit-wise operations which all can be processed using NOR-based operations.

## 3 THERMAL-AWARE DATA ALLOCATION

### 3.1 Thermal Effects of In-Memory Searches

In-memory acceleration for emerging applications involves extensive search operations. To run an HD computing application, the memory required for in-memory search depends on the number of class hypervectors. Since the dimensionality of each hypervector is large, each hypervector is stored across different memory banks to support associative search operations. For example, if each hypervector has a dimensionality of 10,000, a total number of 10 memory banks are required if each memory row has 1024 bit. Encoding different data points can happen in parallel which means multiple pairs of encoding and search modules are used for one application. In this case, the memory requirement for associative searches may increase significantly. Query processing applications may also consume a significant amount of memory for in-memory associative search operations. Associative search operations happen over all data entries available in the dataset, which can easily scale up the amount of required computations.

The power consumption of PIM-enabled memory depends on the number of memory rows supplied by the voltage. The search operation requires the activation of a large number of memory rows resulting in high power consumption. To explore the temperature issues in the 3D PIM architecture, we run an HD computing application for speech recognition. The target application has 26 class hypervectors, each of which has 10,000 dimensions, for each search module which takes up 10 ReRAM banks in the PIM architecture. We use 10 encoding-search bank pairs to balance the size of memory used for storing data points and the computing parallelism. The details of the experimental setup are presented in Section 5.

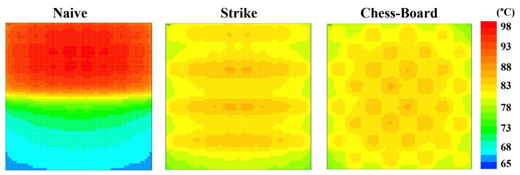

**Figure 3: Temperature distributions of different memory allocation schemes.**

We test three memory allocation schemes as shown in Figure 3. The results show the steady-state temperature distributions of the bottom ReRAM layer, which is the hottest layer in all cases. The results show that "strike" and "chess-board" allocation schemes provide over 8 °C lower peak temperature than "naive" allocation scheme which allocates high power consuming banks together. High temperature may significantly hurt the reliability and endurance of ReRAM device [17, 20]. Specifically, the endurance in terms of total amount of writes can be given by

$$Endurance \approx (\frac{t_w}{t_0})^{\frac{U_F}{U_S}-1} \quad (1)$$

,where $t_w$ is write latency, $t_0$ is a device related constant, $U_F$ is the activation energy for failure mechanisms and $U_S$ is the activation energy of switching mechanism. Based on several previous works [26, 27], $\frac{U_F}{U_S}$ ranges from 2 to 4 for non-volatile memories. $t_w$ has a direct relation with temperature which decreases from 50ns to 16ns when temperature goes from 300 °K (26.85 °C) to 380 °K (106.85 °C) [20]. Such temperature differences cause the endurance of ReRAM devices to range from $4.14 \times 10^8$ to $1.07 \times 10^7$ writes. It is necessary to carefully design the static allocation scheme to optimize temperature behaviors of ReRAM-based PIM architectures.

## 3.2 Optimization Framework

Figure 4 shows the high-level overview of our proposed thermal-aware memory allocation framework. The input of the framework is the application program and PIM architecture characteristics including the memory structure and the operating frequency. Our framework first analyzes the application-specific requirements such as in-memory arithmetic and search operations and the size of the dataset. This information determines the number of banks required for supporting different in-memory operations. Banks supporting in-memory searches are marked as high power-consuming and others are low power-consuming. The power consumption for different banks can be estimated based the static program analysis and the maximum clock rate available in the PIM architecture. We map each operation of the program to corresponding memory banks and assume each bank continuously execute operations at a fixed rate and each operation consumes a fixed amount of energy based on the data size. Once the power consumption of each bank has been decided, our framework explores different allocation schemes to place these banks to different locations inside the PIM architecture to optimize the temperature behavior. The exploration method, which is based on the genetic algorithm, is introduced in the following section. We should note that the overhead of our proposed optimization framework is non-trivial. However, applications we target in this work do not change their characteristics frequently over a long period of time. For example, a PIM architecture may be configured to run image classification using HD computing with a fixed set of class hypervectors for different images generated in the real-world. The operations performed in each bank do not change in this case. Therefore, our proposed exploration algorithm only introduces the overhead when the PIM architecture is configured to run a completely new task.

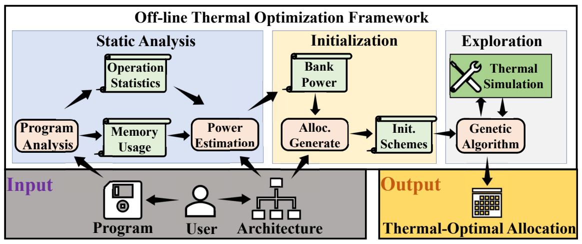

**Figure 4: The framework of thermal-aware data allocation.**

## 3.3 Bank Placement Exploration

Since the energy consumed by search operations is much larger than that consumed by other operations (e.g. normal memory operations and in-memory arithmetic operations), we only focus on allocating power-consuming banks which support in-memory associative searches. In order to efficiently explore the design space, we propose a method based on genetic algorithm [28, 29], which has been successfully applied in the design space exploration for memory controller allocation. However, the number of possible data allocations is extensively large, especially in a 3D stacking architecture. Therefore, we propose a two-phase method to reduce the complexity of exploration.

*3.3.1 Initial States.* In the first phase, we try to generate a set of good initial allocation schemes to start the genetic algorithm for exploration. We utilize the physical characteristics of the 3D stacking structure, where the heat generated in lower layers is much harder to be transferred to the cooling devices than in upper layers. Furthermore, we should also avoid allocating several high power-consuming ReRAM banks together to decrease the power density. One of the initial allocation schemes is "chess-board" which has been shown in Figure 3. The "chess-board" scheme divides memory banks in one layer into two types, where "normal" type is only available for non-search operations and "hot" type can be configured as a high power-consuming bank. For each layer, we start filling "hot" type banks based on the number of search banks assigned to this layer. If all "hot" type banks have been allocated, we start allocating high power-consuming banks to "normal" type banks. We randomly generate different initial states, each of which assigns different numbers of power-consuming banks to different layers. In addition to "chess-board", we also use the "strike" allocation scheme as the initial state and generate different combinations.

*3.3.2 Exploration Method.* The genetic algorithm is used to explore thermal-friendly allocation schemes after the initial population has been generated. We encode each data allocation scheme as a bit vector, which represents the memory location of high/low power consuming ReRAM banks. The "fitness" of the genetic algorithm is the maximum steady temperature across all banks given by our simulation infrastructure. During each "crossover" operation, two-parent solutions are selected from the current population based on "fitness" and a new solution is generated by randomly selecting bits from parents. Then, each newly generated solution is mutated. To utilize the observation that memory banks in upper layers are usually cooler than those in lower layers, we swap a high power consuming memory bank with a low power consuming bank located in the same layer or any of the upper layers. In each generation, we select multiple pairs of parent solutions and each pair generates multiple new solutions to form the next generation. The algorithm completes after a predefined number of generations, or there is no

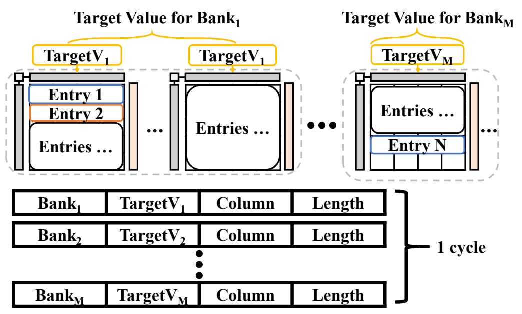
Target Value for Bank$_1$
Target Value for Bank$_M$
TargetV$_1$
TargetV$_1$
TargetV$_M$
Entry 1
Entry 2
Entries ...
Entries ...
Entries ...
Entry N
Bank$_1$ | TargetV$_1$ | Column | Length
Bank$_2$ | TargetV$_2$ | Column | Length
Bank$_M$ | TargetV$_M$ | Column | Length
1 cycle

Figure 5: The original parallel search operation.

significant improvement in "fitness" in the current generation. We test the overhead of running the proposed genetic algorithm in a desktop with a 4-core Intel i7700k processor. The exploration for one application can be finished within 6 hours which is affordable for the off-line optimization.

## 4 DYNAMIC TEMPERATURE MANAGEMENT

Although the static data allocation optimizes the steady-state temperature, the dynamic temperature may still exceed the temperature limit to ensure a reasonable device working situation. As mentioned in Section 1, the conventional Dynamic Thermal Management (DTM) methods, including DVFS, task allocation, and memory remapping, are not suitable for the PIM application with extensive associative searches. Therefore, we propose a configurable in-memory search design as the control knob for fine-grained DTM.

### 4.1 Sub-Table Search

In order to reduce the power consumption of associative search in a certain part of memory, we propose a configurable operation, called *sub-table search*. The original in-memory search activates all rows in the corresponding table in each bank as shown in Figure 5. Banks storing the table are formatted to support in-memory associative search operations and stored in a continuous memory address space. Each search instruction is decoded to a set of bank instructions, which can be processed by corresponding bank controllers. All bank controllers can search data entries simultaneously. The search operation contains a target value which can be decoded to search values in different banks. By searching all rows in different banks simultaneously, a normal search operation can be completed in one cycle. The results of each search are stored in CAM sense amplifiers (SA) in all related banks and can be accessed by following instructions.

Unlike the original search operation, the sub-table search does not activate all memory rows of a table simultaneously. Only a subset of memory rows is activated during one clock cycle which means the power consumption is decreased based on the portion of activated rows. This enables us to only throttle the in-memory search in hot banks without slowing down other operations. To support sub-table search, a search instruction is decoded into multiple groups of bank instructions to complete the search operation in multiple cycles as shown in Figure 6. Each group of bank instructions can be processed by corresponding bank controllers in one cycle. Since the vault controller manages operations to banks in the vault, a control table is added to each vault controller. Each entry indicates the portion of memory rows activated during each cycle when applying an in-memory search in the corresponding memory bank. Since computing divisions during run-time is expensive, we define several levels and add a level table for the controller to look up how many rows should be searched. The level table is

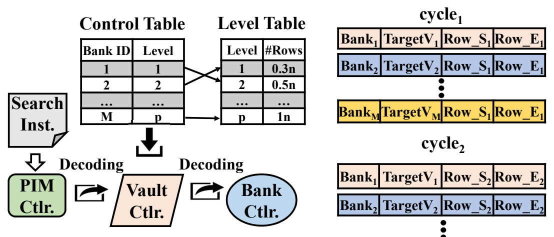
Control Table
Level Table
Bank ID
Level
Level
#Rows
1
1
1
0.3n
2
2
2
0.5n
...
...
...
...
M
p
p
1n
Search Inst.
Decoding
Decoding
PIM Ctlr.
Vault Ctlr.
Bank Ctlr.
cycle$_1$
Bank$_1$ | TargetV$_1$ | Row_S$_1$ | Row_E$_1$
Bank$_2$ | TargetV$_2$ | Row_S$_1$ | Row_E$_1$
Bank$_M$ | TargetV$_M$ | Row_S$_1$ | Row_E$_1$
cycle$_2$
Bank$_1$ | TargetV$_1$ | Row_S$_2$ | Row_E$_2$
Bank$_2$ | TargetV$_2$ | Row_S$_2$ | Row_E$_2$

Figure 6: The proposed sub-table search operation.

**Algorithm 1:** Vault-level DTM based on sub-table search.

```
1: for each $v$ in $Vaults$ do
2:     max_temp=MaxBankTemp(v)
3:     if $max\_temp > hot\_temp$ then
4:         DecreaseLevel($v.b.level$, for $b$ in $v.Banks$)    ▷ Decrease by one level if possible
5:     else if $max\_temp < cool\_temp$ then
6:         IncreaseLevel($v.b.level$, for $b$ in $v.Banks$)    ▷ Increase by one level if possible
7:     end if
8: end for
```

only initiated once when the program allocates a table in memory. The vault controller calculates the number of rows for each level based on predefined portions (0.3, 0.5, ..., 1) shown in the Figure 6. In 4GM HMC design, each vault contains 8 banks and we assume an 8-level setting is used. In this case, each control table consists of 8 `Level` entries, and each `Level` can be represented by a 3-bit value. Each level table contains 8 entries and the number of rows can be represented by a 32-bit value. The area overhead for these tables is 35 Bytes for each vault controller. We should note that the bank-level control table is also applicable to more coarse-grained control mechanisms which configure all banks in a vault with an identical `Level` value.

### 4.2 Fine-grained Temperature Control

During runtime, vault controllers monitor the temperature of all banks in the vault. We assume each memory bank has one digital thermal sensor, which is similar to the design used in previous work [20]. The proposed sub-table search enables us to design a fine-grained DTM which only slows down in-memory searches in specific memory banks without hurting the performance of other operations. The granularity of our proposed DTM is vault, which consists of 8 banks based on the 4GB HMC design [16]. There are two reasons to utilize vault-level control. First, existing 3D memory systems usually divide the control of memory by vault. Vault-level DTM does not require changes to either bank-level or block-level control which may introduce significant overhead. Second, a vault consists of banks throughout all layers in the system which make DTM more effective to control the temperature of different layers. On the contrary, a bank-level DTM may fail to control the temperature when all search-intense banks are near the cooling system. In this case, it is hard to control temperatures of "normal" banks in lower layers by throttling "search" banks in higher layers.

We utilize the step-wise method for our proposed vault-level DTM, as shown in Algorithm 1. If the highest temperature of a vault is higher than a predefined "hot" threshold, the operating state of all banks in the vault decreases by one level. The operating state relates to the portion of rows searched in each cycle. Similarly, if the highest temperature of a vault is lower than a predefined 'cool' threshold, the operating state of all banks in the vault increases by one level. We define 8 levels for configuring the sub-table search operations, which range from 10% to 100%. Since the DTM cannot take effect immediately, the "hot" threshold is set slightly lower than the critical temperature. Furthermore, the "cool" threshold is slightly

Table 1: PIM architecture parameters

| | |
|---|---|
| **PIM Architecture** | HMC 2.1, 4GB, 312.50MHz, ReRAM bank: 16MB, #Banks/Vault: 8, #Vaults: 32 |
| **Memory Parameters** | Resistive memory: 0T-2S, Technology: 45nm |

Table 2: Thermal simulation parameters

| Heat Sink Specification | |
|---|---|
| Convection Capacitance | 140.4 $J/K$ |
| Convection Resistance | 0.1 $K/W$ |
| Conductivity | $400W/mK$ |
| **Stack Dimension** | |
| Heat Sink | $6.0 \times 6.0 \times 0.69cm^3$ |
| Thermal Interface Material Layer (TTSV & TSV-bus) | $50\mu m$ |
| Silicon Layer (Si & TTSV & TSV-bus) | $13.7 \times 13.8 \times 0.1mm^3$ |
| ReRAM Layer (Metal & TTSV & TSV-bus) | $13.7 \times 13.8 \times 0.02mm^3$ |
| D2D Layer (Air & Micro-bumps) | $20\mu m$ |
| Logic Layer (Silicon) | $13.7 \times 13.8 \times 0.1mm^3$ |

Table 3: Application Characteristics

| HD | # Encoding Ops | # Classes | Query | Ops | # Entries |
|---|---|---|---|---|---|
| *FACE* | 617 | 2 | *QUERY1* | MIN/MAX Search | 4M |
| *SPEECH* | 516 | 26 | *QUERY2* | TOP K Search | 6M |
| *PAMAP* | 75 | 12 | *QUERY3* | Exact Search | 8M |
| *MNIST* | 784 | 10 | | | |

lower than the "hot" threshold to prevent frequent fluctuations. The management interval for the proposed DTM is 100ms.

## 5 EXPERIMENTS

### 5.1 Experiment Setup

We implemented a cycle-accurate simulator which simulates application with support of PIM-based instructions. The power consumption and latency of architectural components are calculated based on McPAT [30] and Cacti [31]. For hardware characteristics of PIM logic, we use HSPICE design tool for circuit-level simulations and to calculate energy consumption and performance of all the memory blocks. The energy consumption and performance is also cross-validated using NVSim [32]. The PIM architecture parameters are listed in Table 1. We integrate a widely-used validated thermal simulator, HotSpot [33], with our in-house simulator which generates dynamic power traces. Table 2 shows the detailed HotSpot configuration used in this work. The detailed floorplan of stacking architecture is similar to the previous work which deploys thermal-friendly TSVs to optimize the heat transfer capability [21].

We explore the efficiency of our static and dynamic management policy on the temperature distribution and performance of both machine learning and database applications. For each application, we assume a PIM architecture is fully utilized, thus search operations are performed continuously in memory blocks. For HD computing, we explore the impact of running four popular applications including face recognition (FACE) [34], speech recognition (SPEECH) [35], activity monitoring (PAMAP) [36], and image classification (MNIST) [37]. For query processing, we have tested PIM architecture by running the supported queries in [11]. Specifically, we construct data tables with various sizes and run different sets of query tasks, which are denoted as QUERY1, QUERY2, and QUERY3 in our experiments. Table 3 summaries characteristics of all applications.

### 5.2 Thermal Simulation Validation

We validate simulation results from HotSpot with a commercial electronics cooling simulation software [38]. We model the detailed ReRAM crossbar structure and thermal characteristics of materials based on published works [12, 39]. The materials used for ReRAM and diode are $Ni/HfO_2/PT$ and $Ti/TiO_2/Pt$ respectively. We model the cooling system based on a state-of-the-art

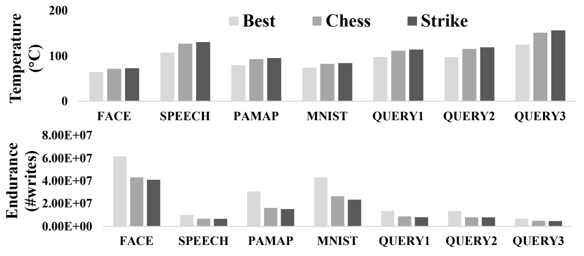

Figure 7: The exploration results of static data allocation.

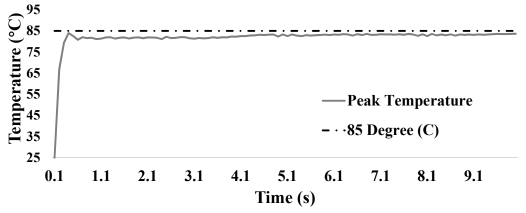

Figure 8: The peak temperature managed by proposed fine-grained DTM when running SPEECH application.

air cooling solution. Specifically, the size of the cooling system is $60mm \times 60mm \times 7mm$, which consists of a $3mm$ subtract and $15 \times 15$ fins. The ambient temperature is set as 25 °C and the heat can be transferred through all surfaces except the bottom. The validation results show that the average error compared to the commercial software is within 7%.

### 5.3 Data Allocation Exploration

Figure 7 shows results of off-line thermal-aware memory allocation for different applications. We only show memory the allocation scheme with the minimum steady-state temperature, and two initial allocation schemes - "Chess" and "Strike". The results show that the best allocation scheme explored by our proposed genetic algorithm can reduce the steady-state temperature at least 15.3 °C on average compared to intuitive allocation schemes. Such results show that carefully allocating different tasks to PIM memories can lead to significant temperature reductions. Based on the temperature result, we estimate the lifetime of each solution based on Equation 1. The value of $\frac{U_F}{U_S}$ is set as 4 and the write latency $t_w$ is estimated by the formula and material-specific characteristics published in the previous work [20]. Figure 7 shows the endurance enhancement provided by temperature reductions. Compared to the "Chess" allocation scheme, our algorithm can extend the endurance by 57.2% on average.

### 5.4 Effect of DTM

As we showed in the static management, the endurance of PIM architecture can vary when running different applications. Dynamic thermal management is a critical method to dynamic thermal behaviors of the PIM architecture. Figure 8 shows the effect of our proposed fine-grained DTM on HD computing running speech recognition task. We set the critical temperature as 85 °C. In order to provide enough time for DTM to control the temperature and avoid frequent fluctuations, we set the control temperature and the recovery temperature as 84 °C and 83 °C respectively. The result in Figure 8 shows that our proposed fine-grained DTM can effectively control the temperature under a predefined critical temperature.

### 5.5 Comparison with Other DTM Methods

Other than controlling the temperature, the proposed DTM is also designed to reduce the performance degradation of coarse-grained

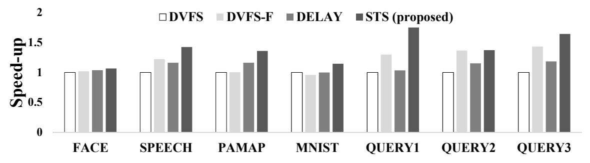

**Figure 9: The performance of different DTMs.**

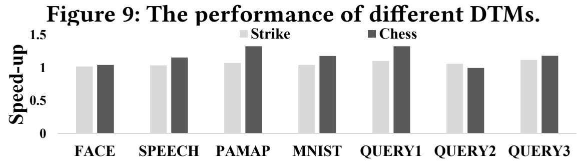

**Figure 10: The performance improvement of the proposed DTM over DVFS-F in different data allocation schemes.**

DTM methods. We compare the performance of our proposed DTM (sub-table-search (STS)) running on the best data allocation scheme with several baselines including a coarse-grained DVFS (*DVFS*) [18], a fine-grained DVFS with the capability of controlling frequency of different vaults (*DVFS-F*), and bank-level operation delaying method (*DELAY*) which reduce the number of memory operations in hot banks [20]. We should note the DELAY method should set a low trigger temperature to ensure the cooling effect (10 °C lower than the critical temperature reported in the prior work [20]). Since applications show significantly different temperatures, we adopt different critical temperature thresholds to make sure all DTM mechanisms are triggered. For example, the critical temperature set for FACE, PAMAP and MNIST is 55 °C, while the critical temperature set for other applications is 85 °C. We use the best data allocation schemes found by our static exploration for all mechanisms.

Figure 9 shows the performance speedup of different DTMs as compared to the coarse-grained DVFS. The results show that our proposed DTM results in 39.7%, 17.6%, and 26.3% speedup as compared to DVFS, F-DVFS, and DELAY respectively. This performance improvement over coarse-grained DVFS (DVFS) mainly comes from removing unnecessary throttling actions in the cool part of memory. In addition, our approach beats the fine-grained DVFS (DVFS-F), since DVFS slows down operations on even on the banks without intensive in-memory search. Even though DELAY exploits a more fine-grained control, its very low trigger threshold degrades the PIM performance significantly.

### 5.6 DTM for Different Data Allocation Schemes

Figure 10 shows the result of running our proposed DTM and the fine-grained DVFS (DVFS-F) in the "Strike" and "Chess" data allocation schemes. The average performance improvements of our proposed method in these two configurations are 7.0% and 17.8% respectively. Our evaluation shows that the proposed approach can beat other methods in various data allocation schemes. The performance benefit provided by our approach depends on the distribution of power consumption.

## 6 CONCLUSION

In this work, we target to solve the temperature issues existing in the emerging PIM applications which require a large amount of high power-consuming in-memory search operations. We extensively explore the design space of the off-line data allocation schemes in a multi-layer PIM architecture to find the best temperature-friendly memory allocation by an efficient and effective algorithm. Furthermore, we propose a configurable in-memory search design and, a fine-grained dynamic thermal management for controlling the dynamic temperature behaviors of the PIM architecture. Based on the experimental result, our proposed method reduces the steady-state temperature and extends the device lifetime significantly. The proposed fine-grained DTM improves the performance of the state-of-the-art DTM method by 17.6%.

## ACKNOWLEDGMENT

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, and also NSF grants #1730158 and #1527034.

## REFERENCES

[1] A. Shafiee *et al.*, “Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.
[2] M. Zhou *et al.*, “Gas: A heterogeneous memory architecture for graph processing,” in *ISLPED*, p. 27, ACM, 2018.
[3] M. Zhou *et al.*, “Gram: graph processing in a reram-based computational memory,” in *ASPDAC*, pp. 591–596, ACM, 2019.
[4] S. Kvatinsky *et al.*, “Magic—memristor-aided logic,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.
[5] M. Imani *et al.*, “Ultra-efficient processing in-memory for data intensive applications,” in *DAC 2017*, p. 6, ACM, 2017.
[6] M. Imani *et al.*, “Floatpim: In-memory acceleration of deep neural network training with high precision,” in *ISCA*, ACM, 2019.
[7] S. Gupta *et al.*, “Felix: Fast and energy-efficient logic in memory,” in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, IEEE, 2018.
[8] M. Imani *et al.*, “Rapidnn: In-memory deep neural network acceleration framework,” *arXiv preprint arXiv:1806.05794*, 2018.
[9] M. Imani *et al.*, “Resistive cam acceleration for tunable approximate computing,” *TETC*, 2016.
[10] Q. Guo *et al.*, “Ac-dimm: associative computing with stt-mram,” *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 189–200, 2013.
[11] M. Imani *et al.*, “Nvquery: Efficient query processing in non-volatile memory,” *TCAD*, 2018.
[12] T. Wu *et al.*, “Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study,” in *IEEE ISSCC*, IEEE, 2018.
[13] M. Imani *et al.*, “Exploring hyperdimensional associative memory,” in *HPCA 2017*, pp. 445–456, IEEE, 2017.
[14] M. Imani *et al.*, “Nngine: Ultra-efficient nearest neighbor accelerator based on in-memory computing,” in *ICRC*, pp. 1–8, IEEE, 2017.
[15] M. Imani *et al.*, “Efficient query processing in crossbar memory,” in *ISLPED*, pp. 1–6, IEEE, 2017.
[16] “Hybrid memory cube specification 2.1.” http://hybridmemorycube.org/specification-v2-download/.
[17] M. V. Beigi and G. Memik, “Thermal-aware optimizations of reram-based neuromorphic computing systems,” in *DAC 2018*, p. 39, ACM, 2018.
[18] D. Brooks and M. Martonosi, “Dynamic thermal management for high-performance microprocessors,” in *HPCA 2001*, pp. 171–182, IEEE, 2001.
[19] A. K. Coskun *et al.*, “Utilizing predictors for efficient thermal management in multiprocessor socs,” *TCAD*, vol. 28, no. 10, pp. 1503–1516, 2009.
[20] M. V. Beigi and G. Memik, “Thor: Thermal-aware optimizations for extending reram lifetime,” in *IPDPS 2018*, pp. 670–679, IEEE, 2018.
[21] A. Agrawal *et al.*, “Xylem: enhancing vertical thermal conduction in 3d processor-memory stacks,” in *MICRO 2017*, pp. 546–559, ACM, 2017.
[22] P. Kanerva, “Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors,” *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.
[23] M. Imani *et al.*, “A framework for collaborative learning in secure high-dimensional space,” in *CLOUD*, pp. 1–6, IEEE, 2019.
[24] A. Haj-Ali *et al.*, “Efficient algorithms for in-memory fixed point multiplication using magic,” in *ISCAS 2018*, pp. 1–5, IEEE, 2018.
[25] J. Ahn *et al.*, “A scalable processing-in-memory accelerator for parallel graph processing,” *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 105–117, 2016.
[26] L. Zhang *et al.*, “Mellow writes: Extending lifetime in resistive memories through selective slow write backs,” in *ISCA*, pp. 519–531, IEEE, 2016.
[27] D. B. Strukov, “Endurance-write-speed tradeoffs in nonvolatile memories,” *Applied Physics A*, vol. 122, no. 4, p. 302, 2016.
[28] D. Abts *et al.*, “Achieving predictable performance through better memory controller placement in many-core cmps,” *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 451–461, 2009.
[29] K.-F. Man *et al.*, “Genetic algorithms: concepts and applications [in engineering design],” *IEEE TIE*, vol. 43, no. 5, pp. 519–534, 1996.
[30] S. Li *et al.*, “Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures,” in *MICRO 2009*, pp. 469–480, IEEE, 2009.
[31] N. Muralimanohar *et al.*, “Cacti 6.0: A tool to model large caches,” *HP laboratories*, pp. 22–31, 2009.
[32] X. Dong *et al.*, “Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory,” in *Emerging Memory Technologies*, pp. 15–50, Springer, 2014.
[33] K. Skadron *et al.*, “Temperature-aware microarchitecture: Modeling and implementation,” *TACO*, vol. 1, no. 1, pp. 94–125, 2004.
[34] Y. Kim *et al.*, “Orchard: Visual object recognition accelerator based on approximate in-memory processing,” in *ICCAD*, pp. 25–32, IEEE, 2017.
[35] “Uci machine learning repository.” http://archive.ics.uci.edu/ml/datasets/ISOLET.
[36] A. Reiss *et al.*, “Introducing a new benchmarked dataset for activity monitoring,” in *ISWC*, pp. 108–109, IEEE, 2012.
[37] Y. LeCun *et al.*, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[38] “Ansys icepack: Electronics cooling simulation.” https://www.ansys.com/products/electronics/ansys-icepak.
[39] P. Sun *et al.*, “Thermal crosstalk in 3-dimensional rram crossbar array,” *Scientific reports*, vol. 5, p. 13504, 2015.