AC-DIMM: Associative Computing with STT-MRAM *

Qing Guo[1] Xiaochen Guo[2] Ravi Patel[2] Engin İpek[1,2] Eby G. Friedman[2]
[2]Department of Electrical and Computer Engineering
[1]Department of Computer Science
University of Rochester
Rochester, NY 14627 USA
[1]{qguo, ipek}@cs.rochester.edu [2]{xiguo, rapatel, friedman}@ece.rochester.edu

ABSTRACT

With technology scaling, on-chip power dissipation and off-chip memory bandwidth have become significant performance bottlenecks in virtually all computer systems, from mobile devices to supercomputers. An effective way of improving performance in the face of bandwidth and power limitations is to rely on associative memory systems. Recent work on a PCM-based, associative TCAM accelerator shows that associative search capability can reduce both off-chip bandwidth demand and overall system energy. Unfortunately, previously proposed resistive TCAM accelerators have limited flexibility: only a restricted (albeit important) class of applications can benefit from a TCAM accelerator, and the implementation is confined to resistive memory technologies with a high dynamic range ($\frac{R_{High}}{R_{Low}}$), such as PCM.

This work proposes AC-DIMM, a flexible, high-performance associative compute engine built on a DDR3-compatible memory module. AC-DIMM addresses the limited flexibility of previous resistive TCAM accelerators by combining two powerful capabilities—associative search and processing in memory. Generality is improved by augmenting a TCAM system with a set of integrated, user programmable microcontrollers that operate directly on search results, and by architecting the system such that key-value pairs can be colocated in the same TCAM row. A new, bit-serial TCAM array is proposed, which enables the system to be implemented using STT-MRAM. AC-DIMM achieves a 4.2× speedup and a 6.5× energy reduction over a conventional RAM-based system on a set of 13 evaluated applications.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Associative memories; B.7.1 [**Integrated circuits**]: Memory technologies

General Terms

Design, Performance

Keywords

Associative computing, TCAM, STT-MRAM

1. INTRODUCTION

Data-intensive workloads play an important role in virtually all computing systems. These workloads often generate significant data movement between the off-chip memory system and the CPU, resulting in considerable power dissipation and a high demand on off-chip memory bandwidth. As CMOS technology scales, both the chip power budget and the available pin bandwidth lag behind significantly, creating a performance bottleneck on data-intensive workloads.

Associative computing using ternary content-addressable memories (TCAMs) has emerged as a promising solution to improve power efficiency and reduce the bandwidth demand of an important class of applications. A TCAM organizes data in *key-value* pairs, and retrieves data by conducting an associative search on the stored keys. This capability has been applied to applications such as data mining [4], network routing [36], text processing [40], search engines [21], and image processing [30].

Conventional TCAM designs using CMOS suffer from low density and high power consumption compared to SRAM and DRAM. Recent work proposes a TCAM DIMM using phase change memory (PCM) [12], which has the potential to scale TCAM capacity to gigabytes; however, the limited functionality constrains the use of the TCAM DIMM to highly search intensive applications.

This paper proposes AC-DIMM, an associative memory system and compute engine that can be readily included in a DDR3 socket. Using spin-torque-transfer magnetoresistive RAM (STT-MRAM), AC-DIMM implements a two-transistor, one-resistor ($2T1R$) cell, which is 4.4× denser than an SRAM based TCAM cell [6]. AC-DIMM enables a new associative programming model, wherein a group of integrated microcontrollers execute user-defined kernels on search results. This flexible functionality allows AC-DIMM to cater to a broad range of applications. On a set of 13 evaluated benchmarks, AC-DIMM achieves an average speedup of 4.2× and an average energy reduction of 6.5× as compared to a conventional RAM based system.

*This work was supported in part by NSF CAREER award CCF-1054179, New York State Office of Science and Technology, and by grants from IBM, Qualcomm, Cisco Systems, and Samsung.

ISCA '13 Tel-Aviv, Israel

2. BACKGROUND AND MOTIVATION

AC-DIMM leverages existing work on associative computing, processing in memory, and resistive memory technologies.

2.1 Associative Memory Systems

Data movement between the processor and main memory is a significant energy and performance bottleneck in modern computers. Associative memory systems, which organize and search data based on explicit key-value pairs, have been recognized as an attractive solution to reduce both the power dissipation and the bandwidth demand as compared to conventional RAM-based systems [12]. An associative memory system can be built in either software or hardware. Software solutions often rely on hash tables, which store and retrieve data by computing a hash function on a set of keys. A hash table provides high storage efficiency on sparse data sets since only useful data are stored; moreover, correlated data can be co-located and retrieved quickly.

Implementing an associative memory system in hardware typically requires a content-addressable memory (CAM) that compares a given search key to a set of stored keys in parallel. Not only does this method avoid the multiple memory accesses that may be needed in a RAM-based hash table, but it also eliminates address computation overheads.

A ternary content-addressable memory (TCAM) is a special type of CAM, which allows both storing and searching with a wildcard in addition to a logic 0 or 1. This flexible pattern matching capability has proven useful in a wide range of applications, including sorting and searching [43], similarity search [44], subset and superset queries [10], decision tree training [19], search engines [21], pattern recognition [14], Huffman encoding [24], and image coding [34]. Despite these advantages, TCAMs have seen limited use in general-purpose computing due to the low density and high power dissipation of existing CMOS-based TCAMs. Goel *et al.* [10] show that a state-of-the-art TCAM device is as fast as SRAM, but the cost-per-bit is 8× higher than SRAM, and the power consumption is 200× more than DRAM. These limitations restrict the commercial use of TCAM to networking applications such as packet classification [22] and IP routing [28].

2.2 STT-MRAM

Resistive memory technologies such as phase change memory (PCM) and STT-MRAM have emerged as viable alternatives to DRAM and SRAM due to their smaller cell area, near-zero leakage power, and enhanced scalability [18]. Resistive memories are generally characterized by high write energy and long write latency. STT-MRAM has lower write energy, shorter write latency, and higher write endurance (virtually unlimited [49]) than other resistive memory technologies [18, 16]. STT-MRAM is a promising alternative for next-generation memory systems: 2Mb [20] and 64Mb [45] STT-MRAM prototypes have been demonstrated, while industry has started to sample 64Mb STT-MRAM chips [29], and has announced that STT-MRAM products will be broadly available in 2013.

The storage element in an STT-MRAM cell is a magnetic tunnel junction (MTJ), which can be modeled as a variable resistor. A typical one-transistor-one-resistor ($1T1R$) STT-MRAM cell is shown in Figure 1. The access transistor is in series with the MTJ. To read the cell, the word line (WL) is asserted and the resistance of the MTJ is sensed. To write the cell, the word line is turned on and the cell is driven by a write current. The direction of the write current determines the value of the bit written to the cell.

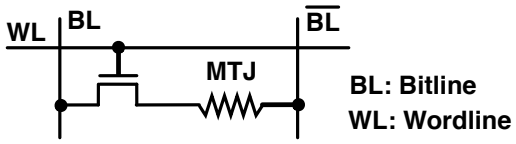


Figure 1: A 1T1R STT-MRAM cell structure.

Matsunaga *et al.* propose STT-MRAM TCAMs with two cell designs: $2T2R$ [26] and $6T2R$ [27]. The $2T2R$ TCAM does bit-serial search, while the $6T2R$ TCAM has a search width of 144 bits due to a pull-up circuit that augments each cell. An STT-MRAM TCAM with high sensing and search speed is reported by Xu *et al.* [46]. Rajendran *et al.* present a $2T2R$ TCAM using PCM [39]. Guo *et al.* explore the design of a $3T3R$ PCM-based TCAM system [12]. Memristor-based CAM and TCAM designs are proposed respectively by Eshraghian *et al.* [8] and Alibart *et al.* [5].

2.3 Processing in Memory

Processing in memory (PIM) aims at reducing data movement overheads by performing computation directly on a memory chip. Elliott *et al.* propose a computational RAM, where the sense amplifiers on a 4Mb DRAM die are augmented with the computational elements of a SIMD pipeline [7]. Gokhale *et al.* develop a PIM chip that can be configured as a conventional memory or as a SIMD processor to accelerate data processing [11]. Oskin *et al.* propose Active Pages [33], wherein microprocessors and reconfigurable logic elements are placed aside DRAM subarrays to process data directly in memory. Different from prior work which combines conventional RAM arrays with logic, AC-DIMM enables content addressability in commodity memories. After launching a search operation, the matched data is automatically selected and processed by a set of local microcontrollers. The desired result is propagated through a reduction tree to a known location and is retrieved by the processor. As such, AC-DIMM significantly reduces the address computation overhead involved in every data access.

2.4 Associative Computing

Significant progress has been made in developing programming models that leverage TCAMs to accelerate a broad range of applications. Potter *et al.* [38] propose an associative programming paradigm (ASC), which enables mapping virtually any RAM-based algorithm to an associative computing framework. Structured data, organized in stacks, queues, trees, matrices, or graphs, are implemented with two-dimensional tables; each entry of a table contains a data item and a descriptor. The data is accessed by launching a search on the descriptor, selecting a matching entry, and retrieving the data for further processing. The AC-DIMM programming model leverages several of the techniques proposed in ASC for general-purpose associative computing.

Guo *et al.* propose a TCAM DIMM which can be modularly integrated into a commodity DDR3 memory system [12]. The proposed $3T3R$ TCAM cell uses phase change memory (PCM), and has an area 20× smaller than its CMOS coun-

terpart when implemented on a standalone (rather than embedded) memory process.

AC-DIMM offers important advantages over the previously proposed TCAM DIMM [12]. First, the cell topology and array organization of AC-DIMM permit a key-value pair to be co-located in the same row, allowing both to be searchable and readable, while requiring fewer devices per cell ($2T1R$). In contrast, the TCAM DIMM (which uses three transistors and three resistors in each cell) does not allow key-value pairs to be co-located; instead, the processor must explicitly read the address of the matching key to locate the corresponding value in a separate DRAM module. Second, AC-DIMM employs a small number of on-die microcontrollers to run user-defined kernels on local search results, which significantly reduces data movement overheads, and allows AC-DIMM to cater to the needs of a much broader range of applications than the TCAM DIMM. Third, the cell topology and array organization of AC-DIMM are applicable to any memory technology that can be used as a RAM, whereas TCAM DIMM requires a high dynamic range (R_{High}/R_{Low}) for its storage elements, which confines the TCAM DIMM to PCM-based cells. A quantitative comparison between AC-DIMM and the TCAM DIMM is presented in Section 8.

3. OVERVIEW

AC-DIMM is a high-performance associative compute engine built on a DDR3-compatible DIMM. Figure 2 illustrates the organization of an AC-DIMM enabled computer system. A multicore processor accesses main memory via an integrated memory controller, which buffers memory requests, schedules memory accesses, and issues DRAM commands over a DDR3 bus. The system supports one or more AC-DIMMs on the memory bus, each comprising an on-DIMM controller and eight associative computing chips (AC-chips). The DIMM controller serves as the interface to the DDR3 bus and manages the DIMM, translating conventional DRAM commands into search operations, and ensuring that DDR3 timing constrains are not violated. The DIMM controller consists of control logic, interface logic, and RAM-based storage (shaded blocks in Figure 2). The RAM block contains control registers for device configuration, a key buffer for buffering the search key, a μCode buffer for buffering microcode, and a result store for buffering search and computation results. To reduce peak power, AC-DIMM adopts a bit-serial search scheme; moreover, only one of the AC-chips can be searched at a time to ensure that the instantaneous power does not exceed the maximum power rating of a standard DDR3 DIMM ($15W$ [31]).

An AC-chip is built from STT-MRAM arrays. A set of specialized microcontrollers, each co-located with a group of four arrays, perform ALU operations on search results, and a reduction tree forwards processed results to the DIMM result store. By mapping part of the physical address space to AC-DIMM, data is made content-addressable and is processed directly by the memory circuit, which significantly reduces data movement and increases energy efficiency.

4. FUNDAMENTAL BUILDING BLOCKS

This section describes the cell structure and circuit design of the AC-DIMM data array. Density is a key factor in memory cell and array design since it affects memory capacity, and plays a dominant role on the length—and thus, the delay and energy—of the wires that are charged and discharged during each access.

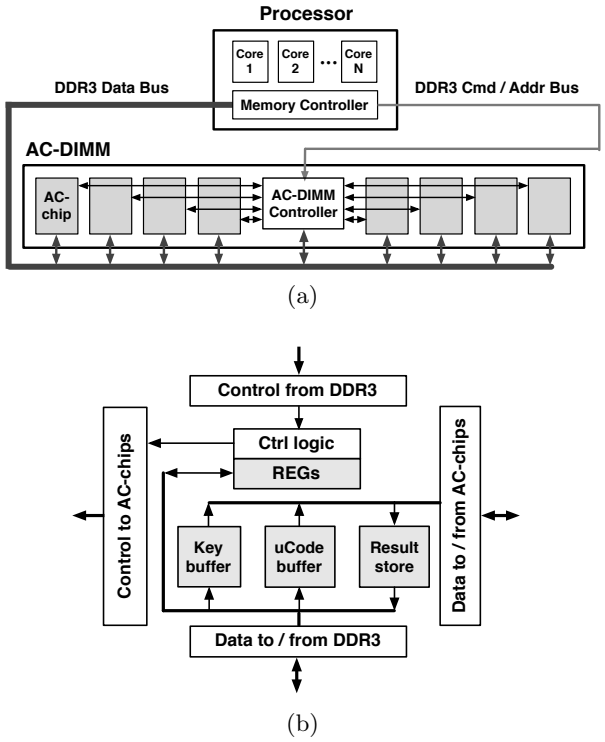
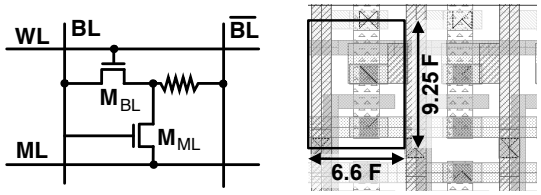


Figure 2: **An example computer system with (a) AC-DIMM and (b) the on-DIMM controller.**



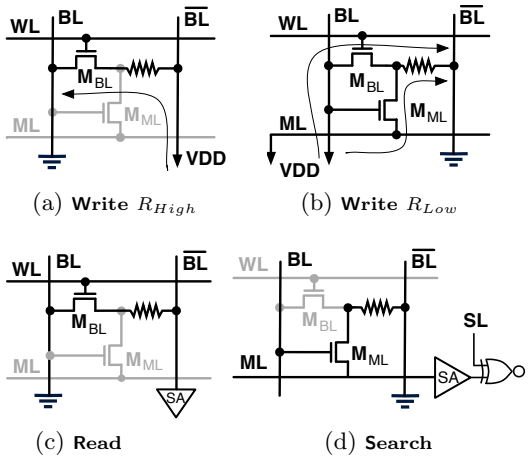
(a) **Cell circuit** (b) **Cell layout**

Figure 3: **Cell topology**

4.1 Cell Topology

An associative memory system requires read, write, and search functionalities from each memory cell. A two-transistor, one-resistor ($2T1R$) cell, shown in Figure 3a, fulfills this requirement with minimal area overhead. Each cell contains a single MTJ that behaves as a storage element. One transistor (M_{BL}), controlled by the wordline (WL), serves as a gating element between two vertical bitlines (BL and $\overline{BL}$), and a second transistor (M_{ML}) connects the MTJ to the horizontal matchline (ML). $\overline{BL}$ and ML are terminated by sense amplifier (SA) circuits.

Writing. Programming a cell requires supplying a write current through the MTJ; the direction of the current determines the written value. To store a logic 0, BL is connected to ground and $\overline{BL}$ is driven to V_{DD}; a write current flows from $\overline{BL}$ to BL, and programs the MTJ to the high resistance (R_{High}) state (Figure 4a). In the reverse direction,



(a) **Write** R_{High} (b) **Write** R_{Low}

(c) **Read** (d) **Search**

Figure 4: **Cell operations**

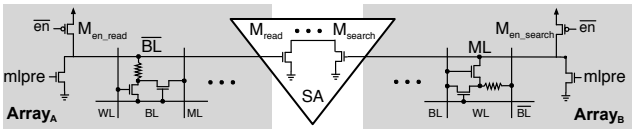


Figure 5: **Sense amplifier circuit**

both ML and BL are driven to V_{DD} and $\overline{BL}$ is connected to ground, which programs the MTJ to the low resistance (R_{Low}) state (Figure 4b). Note that in the latter case, the matchline (ML) actively supplies current to assist the bitline (BL), alleviating the asymmetric write current problem in a conventional $1T1R$ cell [49].

Reading. A read operation is initiated by setting BL to ground and WL to V_{DD}, connecting the MTJ to both bitlines and isolating it from the matchline. A bias circuit then drives $\overline{BL}$. The voltage that appears on $\overline{BL}$ depends on the resistive state of the MTJ (R_{High} or R_{Low}), and is sensed by the sense amplifier (SA) at the end of $\overline{BL}$ (Figures 4c and 5).

Searching. Searching a cell is equivalent to reading the stored bit and comparing it to the search bit (Figure 4d and 5). Specifically, both WL and $\overline{BL}$ are set to ground, BL is connected to V_{DD}, and ML is biased for sensing. The stored data is $XNOR$ed with the searched bit to produce the search result.

Layout. The cell layout, depicted in Figure 3b, has an effective area of 61 F^2 and is based on FreePDK45 to properly model the size of the data array. The cell is made more compact by sharing the bitline, the polysilicon, and the transistor implant contacts between vertically adjacent cells.

4.2 Wildcard Functionality

Wildcard functionality can be achieved by storing a bit and its complement in adjacent columns of a row, as shown in Figure 6. Searching for a 1 or a 0 requires respectively biasing D or $\overline{D}$. A wildcard ("X") is stored in the array by setting both the data bit and its complement to 1. Searching with a wildcard is accomplished by simply skipping the relevant column. Reconfiguration is controlled by user libraries; configuring a group of cells to implement wildcard functionality carries a storage overhead of 2× (i.e., 122F^2) as two bits are required to store every value.

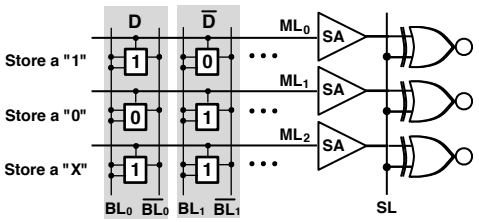


Figure 6: **TCAM reconfiguration**

AC-DIMM can be configured at run-time to work as a RAM, a CAM, or a TCAM. In TCAM mode, the cell is 4.4× more compact than a CMOS TCAM cell [6], but 4.5× larger than the MTJ based $3T3R$ cell (27F^2) presented in prior work on TCAM DIMM [12], which assumes ITRS parameters for the device area. ITRS assumes a standalone memory process where spacing can be more aggressively optimized due to regular patterning within the data arrays. The aggressive design rules for standalone memory processes are not publicly available; the results presented here are representative of an embedded STT-MRAM logic process. For comparison, the layout of the $3T3R$ cell presented in [12] occupies a cell area of 164F^2 when implemented using FreePDK45.

4.3 Bit Serial Operation

Existing TCAM circuits generally favor large search widths (72-576 bits [10]) to fulfill system throughput requirements. However, the low dynamic range (R_{High}/R_{Low}) of STT-MRAM prevents the search width from scaling beyond a few bits [12]. AC-DIMM implements a *bit-serial* search scheme, wherein a search operation progresses iteratively, first sensing the cell on the initial column, and progressively searching column by column across the data array. The results presented in Section 8 indicate that bit-serial operation can improve performance and energy-efficiency when search keys are short.

Supporting bit-serial operation requires additional logic within the memory array. Each row is augmented by sense amplifier logic, helper flip-flop logic, and multi-match resolution logic (Figure 7). The sense amplifier logic resolves a single bit of the search result (by $XNOR$ing the stored bit and the search bit), which is ANDed with the previous result (stored in FF1). The helper flip-flop (FF2) buffers the current search result, where a logic 1 represents a match.

A multi-match resolution (MMR) circuit makes it possible to read out matching rows successively at the end of a search, without requiring any additional decoding. The MMR is based on a token passing design [9] and serves to select the first match within a data array. Upon conclusion of a search, the value stored in the helper flip-flop is passed to a match buffer. The MMR operates by precharging a chain of pass transistors and discharging the chain up to the first matching row (i.e., giving the token to the first match). Subsequently, the match buffer for the row is cleared and the MMR passes the token to the next matching row.

5. ARCHITECTURE

Architecting a high-performance associative memory system requires striking a careful balance between two important design metrics. First, memory capacity must be suf-

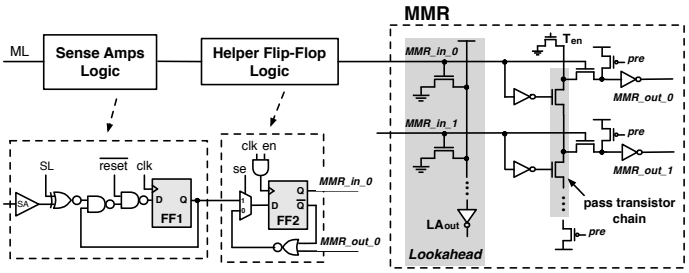
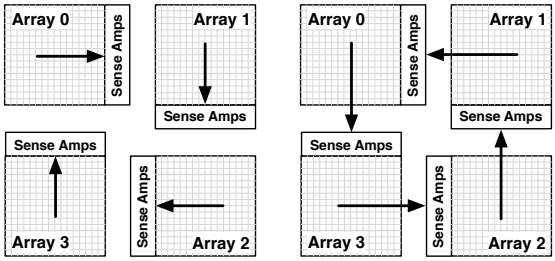


Figure 7: **Embedded row logic**



(a) **Search organization** (b) **Read organization**

Figure 9: **Illustration of sharing the sense amplifiers.**

ficiently large to accommodate most data sets; second, the associative memory system must support a sufficiently diverse set of operations on the search results (e.g., sum, min, max) to satisfy the demands of different applications. For a fixed chip area and power budget, memory capacity and rich functionality are often at odds: adding more compute capability to a memory circuit requires precious area, and can consume considerable power.

5.1 Array Organization

AC-DIMM allows each memory row to be searched, read, and written—a key capability absent from prior work on TCAM DIMM [12]. A row is read and written through vertical bitlines and is searched through horizontal matchlines (Figure 8); a row decoder, bitline drivers, and bitline sense amplifiers enable reads and writes; bitline drivers, matchline sense amplifiers, and the helper flip-flop (Section 4.3) assist the bit-serial search.

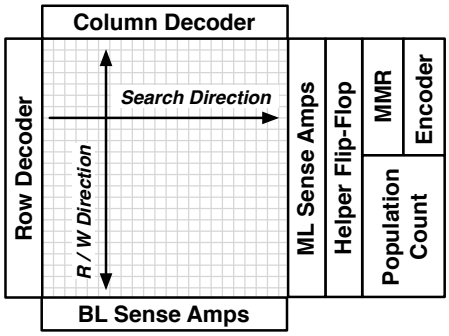


Figure 8: **Array organization.**

The helper flip-flop outputs the word match results after each stored bit is iteratively compared against the corresponding bit of the search word. The population count logic, the multi-match resolution (MMR) circuit, and the encoder are used in processing match results. The population count logic computes the number of matches. The MMR circuit selects the first match among all matches, and outputs a one-hot vector, in which only the row corresponding to the first match contains a 1 (Section 4.3). The output of the MMR circuit can be directly used to drive the wordline to read or write the selected row in the following cycle, which eliminates the need to use a decoder to select the wordline. The outputs of the MMR circuit also connect to the encoder to generate the encoded index of the selected matching row; hence, the MMR circuit and the binary encoder together function as a priority encoder. Partitioning priority encoder functionality between the MMR and the binary encoder circuits simplifies the circuit design process, and enables efficient read-after-search and write-after-search operations.

Note that in Figure 8 there are two sets of sense amplifiers: one set connects to the bitlines and another set to the matchlines. The structure and functionality of the matchline and bitline sense amplifiers are identical. It is possible to eliminate one set of sense amplifiers by sharing sense amplifiers across data arrays, thereby reducing the area overhead of the sense amplifier blocks. The cycle time of a bit-serial search is more critical than the cycle time of a read—a 32-bit search requires 32 cycles, whereas a 32-bit read requires only one cycle. The matchline sense amplifiers should therefore be close to the matchline, since the additional latency to sense the bitline signals from the sense amplifiers of a neighboring data array is tolerable. An efficient sense amplifier sharing scheme can exploit the orthogonal read and search directions by rotating the arrays and sharing the sense amplifiers: the matchline sense amplifers for one array are used as the bitline sense amplifers of another array. Figure 9 shows an example: array 1 is rotated 90^o with respect to array 0; array 2 is rotated another 90^o with respect to array 1; and array 3 is again rotated 90^o. On a search (Figure 9a), the sense amplifiers adjacent to each array are used; on a read (Figure 9b), the sense amplifiers of a nearest neighbor are leveraged. To enable sharing, all four arrays are restricted to perform the same type of operation (read or search) at any time.

5.2 Microcontroller

AC-DIMM provides a user program flexible control over the reduction operations to be performed on the search results by placing a specialized microcontroller at the center of the four arrays. The microcontroller implements a simple RISC ISA with 16-bit instructions (Table 1). The instruction formats are listed in Figure 10.

ALU	ADD, ADDU, SUB, SUBU, MULT, MULTU, DIV, DIVU, MOVE, LI, NEG, AND, OR, NOT, XOR, NOP
Control	BNEZ, BLEZ, BGTZ, J
Memory	LB, LBU, LH, LHU, LW, SB, SH, SW
Special	NEXT

Table 1: **Microcontroller instructions.**

Figure 10 also demonstrates the four-stage microcontroller pipeline. There are 32 registers: 16 are general purpose, six are special purpose, and ten are data registers pre-loaded with immediates. The six special purpose registers include

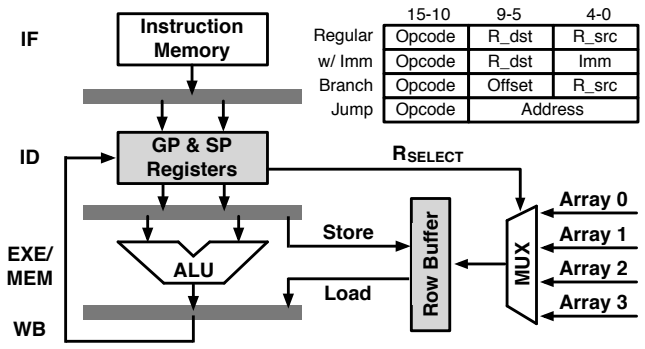


Figure 10: **Illustrations of the microcontroller and the instruction formats.**

R_{HI} and R_{LO} for multiplication and division, R_{SELECT} for selecting one of the four arrays, R_{OUT} for communicating to the upper level reduction tree node, R_{COUNT} for holding the match count of the selected array, and R_{INDEX} for holding the encoded index of the selected matching row. Both memory and arithmetic operations are handled in the execution stage. Memory instructions operate on the row buffer of the selected array. The row buffer is addressed implicitly, eliminating address computation. The special instruction NEXT signals the MMR circuit to reset the current matching row to a mismatch. The MMR circuit then selects the next matching row, which allows the microcontroller to iteratively read or write the set of matching rows.

5.3 Reduction Tree

After the local computation at the microcontroller completes, the reduction tree computes three outputs: the sum of all local microcontroller results, the total match count, and the index of the first matching row. While other operations could be incorporated into the reduction tree, these three operations are sufficient to accelerate a broad range of applications considered in the evaluation.

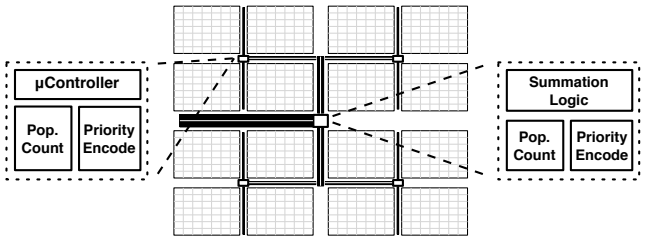


Figure 11: **Illustration of the reduction tree.**

As shown in Figure 11, every group of four nodes share a reduction tree node that outputs its results to the next higher level of the tree. Arithmetic overflows may occur in the microcontrollers or the summation logic. In such a case, the on-DIMM controller is notified and a flag register is set. In the case of a division by zero, microcontrollers and summation logic are aborted and diagnostic information (i.e., the type of exception), the current μCode pointer, and the memory location pointer are stored in the result store. AC-DIMM relies on the user library to handle these exceptions.

5.4 System Interface

AC-DIMM can be modularly included in a DDR3 memory system without modifying the processor die or the motherboard. The on-DIMM controller serves as the interface to the system memory bus. It includes a RAM block containing control registers, a key buffer, a μCode buffer, and a result store (Figure 2) that the processor can access. The controller manages all AC-DIMM operations and ensures that the DDR3 timing constrains are not violated.

Memory Allocation and Deallocation. A *create* function allocates a block of memory from the (associative) physical address space. To simplify cache design and to avoid cache flushes, memory requests sent from the processor to the associative memory are made uncacheable[1]. A *destroy* function reclaims allocated memory for future use.

The basic commands provided to the processor are read, write, and search. Read and write are ordinary load and store instructions except that the destination address is a memory mapped location in an uncacheable page. A search is initiated by a set of stores. A search command sends a control vector to the AC-DIMM controller; the vector specifies a pointer to the search key, the search key length, the region of memory to be searched, and the desired reduction operation. The key is stored in the key buffer prior to a search. Upon receipt of the search command, the AC-DIMM controller reads the search key out of the key buffer, and broadcasts it to the arrays in the selected region. The controller iteratively issues a number of bit-serial search operations, and the desired results are forwarded to the result store. A search command outputs the following results: 1) the number of matches, 2) the index of the first matching row, and 3) a result computed by a user-defined kernel executing on the local microcontrollers.

DDR3 Interface. Similar to the TCAM DIMM [12], AC-DIMM can be modularly included in a DDR3 memory system. A typical DDR3 transaction is not sufficiently long to ensure the completion of every AC-DIMM operation. Rather than pushing the memory logic to run faster (which requires additional power and area overhead), we use the *dummy write* scheme proposed by Guo et al [12]. A dummy write is effectively an idle cycle on the memory bus generated by writing a $NULL$ value to a known location on the on-DIMM controller. The following process describes how a dummy write is used.

The processor starts an AC-DIMM compute session with a *query* (Section 6), and subsequently issues a read from the status register of the on-DIMM controller, which records the number of matching rows in the array with the highest match count. The number of matches and the length of μCode are used to compute the number of cycles remaining before the desired result is produced.[2] The processor then issues an appropriate number of *dummy writes* before reading the search results.

The peripheral circuitry of the array, the microcontroller, and the reduction tree are all pipelined, which significantly improves the processing throughput and makes the overhead of the *dummy writes* tolerable.

[1]"Uncacheable" is a page attribute supported by Intel since i386 [17] and by AMD since AMD64 [3], which prevents the memory requests sent to the page from entering the cache subsystem, and memory requests from being reordered.

[2]This requires the μCode to be written such that its worst case execution time can be statically inferred.

To reduce the programming effort, a user-level library is provided to facilitate mapping applications to the AC-DIMM programming framework. Bubbles are automatically inserted to obey the DDR3 timing constraints.

6. PROGRAMMING MODEL

AC-DIMM combines content addressability and processing-in-memory. The cell structure and array organization allow data to be co-located with a descriptor, which is used for retrieving the data. On a match, the microcontrollers (located with every group of four arrays) are signaled to run user-defined μCode on the search results and generate partial results. These partial results propagate to the on-DIMM result store via a reduction tree; in the case of multiple matches, the microcontroller iteratively processes each matching entry. Notably, the wordline of the matching row is automatically selected by the MMR circuit; hence, no address calculation is required. When all of the microcontrollers finish, the on-DIMM controller collects and combines the partial results. The processor issues a read command to return the final result.

6.1 Partitioning

Realizing the full potential of AC-DIMM requires the workload to be partitioned between the processor and AC-DIMM. The AC-DIMM computational model exhibits the best performance when data level parallelism is present. The current version of AC-DIMM only supports integer computation; nevertheless, on a floating-point application, AC-DIMM can gather and organize the data for the processor.

6.2 Data Structures

The data structures used in AC-DIMM are motivated by the associative computing paradigm proposed by Potter [38]. Structured data organized in stacks, queues, matrices, trees, and graphs are converted into a tabular form. Each entry of the table contains a key-value pair, where the key is a user-defined descriptor. In certain cases, the content itself may be a descriptor, and is searched. String Match [40], for instance, aims at determining if a query string is present anywhere in a string dataset. The AC-DIMM implementation of String Match uses the string itself as the search key, and reads the population count to determine the number of matches for a query.

6.3 Implementation

The AC-DIMM programming model defines a query-compute-response framework, where *query* and *response* are two primitives.

To initiate a compute session, the processor issues a *query* to the AC-DIMM controller, which assembles the search key and broadcasts the key to a set of arrays. Once the search finishes, the local microcontrollers are signaled to execute a set of user-defined operations and iteratively process each matching row. Partial results generated by the microcontrollers are propagated through a reduction tree to produce the final result. The processor retrieves the *response* from the memory-mapped result store (Figure 12).

User-level library functions largely hide the operational details from the user, reducing the programming effort and improving the maintainability of the code. However, μCode is application-specific and must be written by the user. We claim this effort is tolerable since the control flow is automatically handled by the MMR circuit. Example applications described in Section 7.3 show that the μCode for most applications is both intuitive and short.

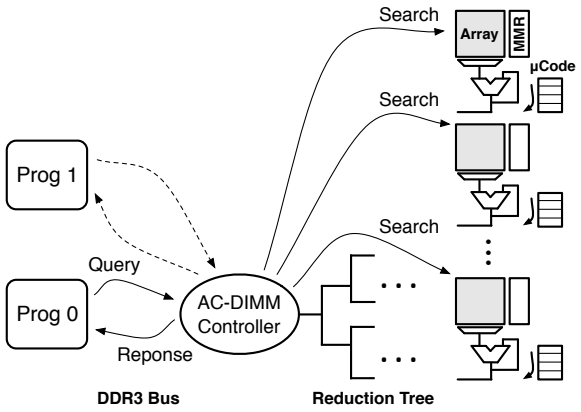


Figure 12: **An illustration of the proposed AC-DIMM programming model.**

6.4 Application Mapping

Case studies on example benchmark applications are presented in this section. Mapping an application to the AC-DIMM programming model is a three step process: 1) defining the key-value data structure, 2) partitioning the workload between the processor and AC-DIMM, and 3) designing the μCode.

Apriori [32] is an association-rule mining algorithm that finds large itemsets in a set of transactions. The *key* is a bit vector which records the presence of each item in a transaction. No *value* accompanies the key.

The processor stores the keys in the AC-DIMM data array. For each candidate itemset, the *query* function searches the corresponding columns marked by the search key. The *response* function returns the number of matches (population count). No μCode is required for this application.

Matrix Multiplication [40] computes the product of two input matrices. The *key* comprises a bit string representing the matrix address, row index, and column index. The *value* is the corresponding matrix entry.

The processor stores the key-value pairs in the AC-DIMM data array. For each entry in the result matrix, the processor *queries* the *row* and *column* indices. The μCode reads the corresponding operands from the matching rows in each input matrix array and computes the product. The singleton products are summed by the reduction tree adders to produce the final result, which is returned by the *response* function.

Word Count [40] calculates the number of distinct words in a text file. A data element has two *keys*: the first key, a file ID, denotes which text file the word belongs to; the second *key* and the *value* are the word itself.

The processor parses the input text and stores the data elements in the AC-DIMM data array. The *query* function searches for a word and the *response* function returns the number of matches. To fetch the following search key, the processor disables all previous matching rows (by signaling the "enable" flip-flop in each matching entry [12]), and *queries* on the file ID. The *response* function uses the priority index to read the value from the first matching entry

Core	8 cores, 4.0 GHz, 4-issue
Functional units	Int/FP/Ld/St/Br units 2/2/2/2/2, Int/FP Mult 1/1
IQ, LSQ, ROB size	IssueQ 32, LoadQ/StoreQ 24/24, ROB 96
Physical registers	Int/FP 96/96
Branch predictor	Hybrid, local/global/meta 2K/2K/8K, 512-entry direct-mapped BTB, 32-entry RAS
IL1 cache (per core)	32KB, direct-mapped, 32B block, 2-cycle hit time
DL1 cache (per core)	32KB, 4-way, LRU, 32B block, hit/miss delay 3/3, MESI protocol
L2 cache (shared)	4MB, 8-way, LRU, 64B block, 24-cycle hit time
Memory controller	4-channel, 64-entry queue, FR-FCFS, page interleaving
AC-DIMM Subsystem	256MB AC-DIMM, 8 chips per DIMM, DDR3-1067 MHz
Timing (DRAM cycles) [31]	tRCD: 7, tCL: 7, tWL: 6, tCCD: 4, tWTR: 4, tWR: 8, tRTP: 4, tRP: 7, tRRD: 4, tRAS: 20, tRC: 27, tBURST: 4, tFAW: 20

Table 2: **System architecture core parameters.**

Application	Benchmark	Category	Problem size	Search key (width)	AC-DIMM computation
Apriori	MineBench	Association rule mining	95,554 transactions, 1000 items	Itemset (2-5 bits)	Count the matches in the transaction set
Histogram	Phoenix	Image processing	6816 × 5112 BMP	Pixel values (8 bits)	Count the matches in the image
Reverse Index	Phoenix	Text processing	HTML files, 1GB	URLs (64-1024 bits)	Find the matched documents IDs
String Match	Phoenix	Text processing	10MB key file, 0.5KB encrypted file	Encrypted keys in the encrypted file (128 bits)	Find the match
Word Count	Phoenix	Text processing	10MB text file	Words (8-128 bits)	Count the word matches
Matrix Multiply	Phoenix	Scientific computing	500 × 500 matrices	Matrix indices (32 bits)	Compute inner product on any selected two vectors
Kmeans	Phoenix	Clustering	10000 3D points 20 means	Data set ID, cluster IDs (8 bits)	Compute point-mean distance and update the cluster center
PCA	Phoenix	Statistical learning	500 × 500 matrix	Matrix indices (8 bits)	Compute mean matrix and covariance matrix
Linear Regression	Phoenix	Optimization	10MB points	Vector indices (8 bits)	Compute sum and inner product of vectors
BitCount	MiBench	Automotive	75000 bit vectors	Bit positions (1 bit)	Count the 1s in the data set
Susan	MiBench	Automotive	384 × 288 PGM	Pixel indices (32 bits)	Compute the average brightness
Dijskstra	MiBench	Network	100 vertex graph	Vertex ID (32 bits)	Find the adjacent vertex
Vortex	SPEC CINT2000	Associative database	3 inter-related DB, 200MB	Item ID (32 bits)	Find the matched entry

Table 3: **Workloads used with the AC-DIMM system.**

(which is the next distinct word to be searched). No μCode is needed in this application.

Reverse Index [40] creates a reverse indexing table from URLs to HTML files that contain those URLs. Each *key* is a URL, and the corresponding *value* is the document ID that contains the URL.

The processor first parses the HTML files, extracts the URLs, and stores the key-value pairs in the AC-DIMM data arrays. Next, the processor *queries* AC-DIMM using the URL as the search key. For each match, the μCode reads the document ID and forwards it to the on-DIMM result store. Finally, the *response* function returns all the document IDs found by the *query*.

Kmeans [40] is a clustering algorithm that groups a set of input data points into clusters. It assigns each data point a cluster ID and outputs the generated cluster centers. A data point has two *keys*: a data set ID and a cluster ID denoting which cluster the data point belongs to. The data point itself is the *value*.

The algorithm proceeds in two phases. The processor first sends the microcontrollers the cluster centers. For each cluster center, the processor issues a *query* on a data set ID. The μCode reads each matching point, computes its distance to the cluster center, and updates the cluster ID of that point. No *response* function is needed in the first phase. Second, the processor *queries* each of the cluster IDs. For each search, the data points that belong to the cluster are selected. The μCode sums up the matching points. The *response* function returns the sum and the number of matches (population count). The processor computes the new cluster centers and decides whether to perform one more iteration of the main loop.

7. EXPERIMENTAL SETUP

Circuit, architecture, and application level simulations have been conducted to quantify area, energy, and performance (Section 8). Tools, parameters, and assumptions used for the experiments are described in this section for each level of the simulations.

7.1 Circuits and Synthesis

Cadence SPECTRE circuit simulations have been conducted for all constituent parts of the data array including the cell, the embedded row logic, and the wordline and bitline driver circuitry. The MTJ element is modeled with a simple bias dependent tunneling magnetoresistance (TMR) model[3] (Equation 1), in conjunction with ITRS parameters for the 22nm node [18]. Process-voltage-temperature (PVT) variations may cause TMR degradation of an MTJ. We consider a 10% reduction in the R_{High} value and a 10% increase in the R_{Low} value, which produces the worst case results.

[3]TMR is a measure of the change in the resistance of the MTJ between the high and low states: $TMR = (R_{high} - R_{Low})/R_{Low}$. Physical device studies show that R_{High} degrades as a function of the applied bias [35]. The bias dependent TMR model is a simple curve fitting approach to emulate the voltage dependence of the TMR, which has been used in published compact device models [25].

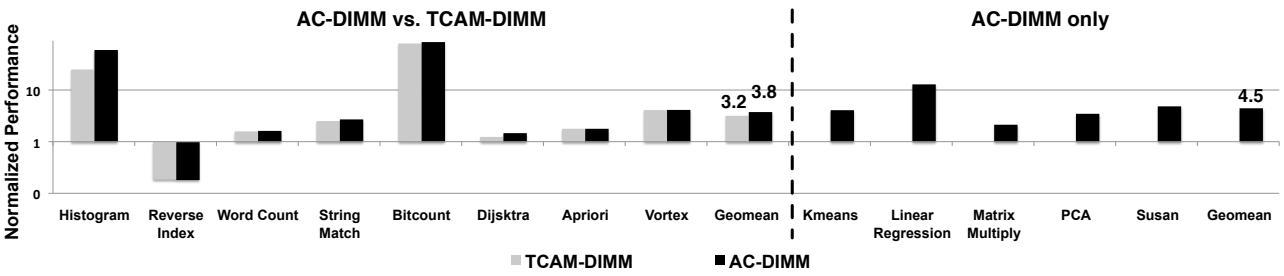


Figure 13: **AC-DIMM and TCAM DIMM performance speedup normalized to the DRAM-based system.**

$$TMR(V_{MTJ}) = \frac{TMR_0}{1 + \frac{V_{MTJ}^2}{V_h^2}}. \quad (1)$$

The CMOS transistor model is based on the predictive technology model (PTM) for the 22nm node [48]. Low threshold CMOS devices are modeled by scaling the device thresholds and velocity saturation parameters, in a manner consistent with the device models provided in FreePDK45 [2]. The circuit layout includes the array cell, as well as the logic gates within the embedded row logic. The embedded row logic circuitry is pitch-matched to the memory cell. The layout geometry is scaled to the 22nm node and pertinent interconnect parasitic impedances are extracted using the PTM interconnect model and back-annotated into the circuit simulation.

The area, energy, and delay of the decoders, population count circuits, encoders, summation logic, and local microcontrollers are estimated from RTL code and design compiler [1] synthesis. FreePDK45 [2] is the synthesis library. The results are scaled from 45nm technology to 22nm technology by parameters (Table 4) generated by SPECTRE circuit simulations.

	FO4 Delay (ps)	FO4 Leakage (nW)	Voltage (V)
45nm	11.8	2.45	1.0
22nm	6.5	5.68	0.8

Table 4: **Technology scaling parameters.**

The interconnect (H-tree connecting memory arrays) area, energy, and delay are calculated using FreePDK45 metal layer parameters [2] and the PTM [48] interconnect model. The intermediate metal layer is chosen for the H-tree.

7.2 Architecture

The SESC [41] simulator was augmented to model an eight-core computer system. A 256MB AC-DIMM (eight AC-chips per DIMM) system connects to the memory controller through a DDR3-1067 bus. The memory controller uses page-interleaving address mapping [47] and the FR-FCFS scheduling method [42]. Core architectural parameters are listed in Table 2. CPU energy is evaluated using McPAT [23].

7.3 Applications

AC-DIMM can be utilized by a large number of applications including data mining, clustering, text processing, scientific computing, and image processing. We compose a benchmark suite that represents a wide range of applications from NU-MineBench [37], Phoenix [40], MiBench [13], and SPEC CINT2000 [15]. AC-DIMM is evaluated with the single-threaded version of each benchmark, while the evaluated baseline runs the parallel applications (Phoenix benchmarks and Nu-MineBench) with eight threads. Detailed workload parameters are listed in Table 3.

8. EVALUATION

This section describes architectural and circuit-level evaluations of AC-DIMM. Performance, energy, and area are reported. The results are compared to a DRAM-based system and the previous work on TCAM DIMM [12].

8.1 Performance

Over a set of eight workloads, both AC-DIMM and TCAM DIMM outperform the DRAM-based system by an average speedup of 3.8× and 3.2×, respectively (Figure 13). (The remaining five workloads are not portable to TCAM DIMM due to its limited flexibility; these workloads are shown on the right of Figure 13.) The performance improvement comes from using a content-addressable memory (in this case, AC-DIMM and TCAM DIMM); on a search, the search key is simultaneously compared against all of the stored keys and the matching one is returned. This approach surpasses any software algorithm since retrieving data out of the memory is not required, which not only improves the performance, but also eliminates unnecessary power consumption and bus traffic.

AC-DIMM presents an additional 19% performance improvement over the TCAM DIMM. Note that the STT-MRAM based AC-DIMM searches in a bit-serial fashion (Section 4.3), whereas the TCAM DIMM using PCM has a 128-bit search width. As compared to a wide search, bit-serial search results in lower peak power (Figure 14b), and allows AC-DIMM to implement more aggressive (and hence faster) peripheral circuitry than the TCAM DIMM. Figure 14a shows the delay of a search followed by a reduction tree operation with respect to the search width. For both reduction tree operations (i.e., population count and priority index), AC-DIMM exhibits a lower delay when the search width is less than 32 bits (Table 3).

Notice that the performance of Reverse Index deteriorates in both AC-DIMM and TCAM DIMM implementations. Reverse Index spends considerable execution time on text parsing. The irregular control flow (regular expression processing) does not lend an efficient mapping to an associative memory and runs on the processor instead (Section 7.3).

Beyond associative memory capability, AC-DIMM is able

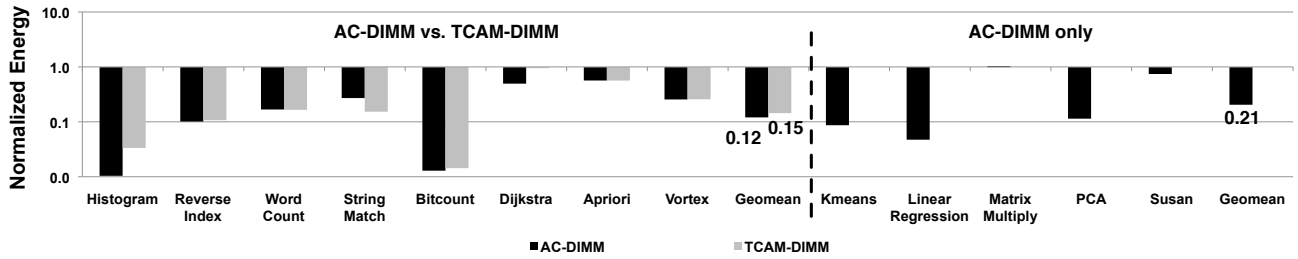
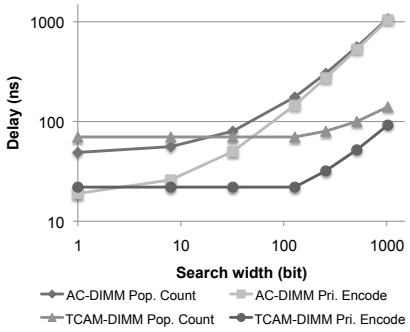
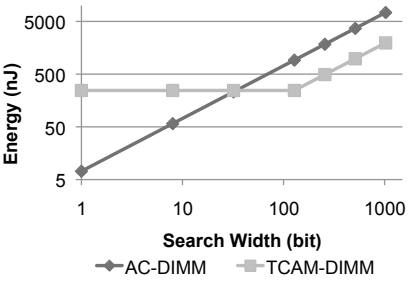


Figure 15: **AC-DIMM and TCAM DIMM energy normalized to the DRAM baseline.**



(a) Delay



(b) Energy

Figure 14: **Delay and energy comparison**

to apply a rich set of user-defined operations to the search results (i.e., matches), and can process the required results *in situ*. Figure 13 shows performance over a set of simulated applications. AC-DIMM outperforms the eight-threaded RAM-based multicore system by an average of 4.5×. As data intensive applications often suffer from insufficient off-chip memory bandwidth, these applications naturally benefit from the processing-in-memory capability of AC-DIMM. In contrast, these applications cannot benefit from the TCAM DIMM due to insufficient functionality. The two operations provided by the TCAM DIMM (i.e., population count and priority index) see limited use in these workloads.

AC-DIMM allows a key-value pair to be co-located within a row, which allows the value to be read and used immediately after a match. The TCAM DIMM array architecture does not provide a separate read port such that data can be read in the same way it is written. This method generates extra overhead: when a search operation finishes, the processor must explicitly use the matching index to fetch the data from a separate DRAM module, resulting in additional memory accesses.

8.2 Energy

AC-DIMM enables local computation by reducing data movement between the processor and the memory, thereby reducing the total system energy. To operate within the power budget of a DDR3 DIMM, bit-serial search is employed. However, as the search width increases, bit serial search becomes less energy efficient than the 128-bit search mechanism used in the TCAM DIMM design. This characteristic is due to the additional energy consumed by repetitively activating the matchline and the bit-serial match resolver (Figure 14b). A comparison of the system energy is shown in Figure 15, which includes the eight applications that are portable to both AC-DIMM and TCAM DIMM, and the five applications which are only portable to AC-DIMM (Section 7). AC-DIMM saves more energy than the TCAM DIMM in Histogram, Reverse Index, Bitcount, and Dijkstra, because the associative key and data are co-located in AC-DIMM. AC-DIMM eliminates the additional RAM accesses required in a TCAM DIMM system. String Match consumes more energy in the AC-DIMM system than in the TCAM DIMM because the search width in this application is 128 bits; hence, searching consumes more energy in AC-DIMM. The five applications which are exclusively portable to AC-DIMM exhibit energy savings over the baseline with the exception of Matrix Multiply. Matrix multiplication is highly computation intensive. Running Matrix Multiply on AC-DIMM consumes the same amount of energy as on a multi-core processor.

8.3 Area

The area breakdown of the proposed AC-DIMM is shown in Figure 16. The area efficiency[4] is 32%, which is compara-

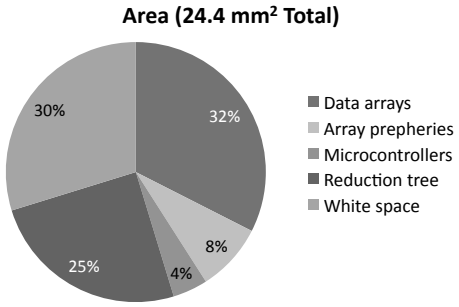


Figure 16: **Area breakdown.**

[4]The area efficiency of a memory circuit is the ratio of the total area occupied by cells to the total chip area. According to ITRS [18], the area efficiency of DRAM is 55%.

ble to the area efficiency of the TCAM DIMM (37%). The white space is reserved for clock distribution, power distribution, and decoupling capacitors.

9. CONCLUSIONS

This paper presents AC-DIMM, a high-performance, energy-efficient associative memory DIMM that can be modularly included in a DDR3-compatible system. AC-DIMM uses a novel $2T1R$ STT-MRAM cell, which is 4.4× denser than a CMOS TCAM cell. When implemented in an embedded STT-MRAM process, the cell topology is applicable to any memory technology that can be used as a RAM. AC-DIMM broadens the scope of associative memory systems over existing approaches by allowing a key-value pair to be co-located in the same row, and by employing integrated microcontrollers to execute user-defined operations on search results. The end result is a high-performance, energy-efficient solution that successfully combines associative search and processing in memory capabilities.

10. REFERENCES

[1] *Design Compiler Command-Line Interface Guide*. http://www.synopsys.com/.

[2] Free PDK 45nm open-access based PDK for the 45nm technology node. http://www.eda.ncsu.edu/wiki/FreePDK.

[3] Advanced Micro Devices, Inc. *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, 2010.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th Very Large Databases Conference*, Santioago de Chile, Chile, Sept. 1994.

[5] F. Alibart, T. Sherwood, and D. Strukov. Hybrid CMOS/nanodevice circuits for high throughput pattern matching applications. In *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, June 2011.

[6] I. Arsovski, T. Chandler, and A. Sheikholeslami. A ternary content-addressable memory (TCAM) based on 4T static storage and including a current-race sensing scheme. *Solid-State Circuits, Journal of*, 38(1):155 – 158, Jan. 2003.

[7] D. Elliott, W. Snelgrove, and M. Stumm. Computational RAM: A memory-SIMD hybrid and its application to DSP. In *Custom Integrated Circuits Conference, 1992., Proceedings of the IEEE 1992*, pages 30.6.1 –30.6.4, May 1992.

[8] K. Eshraghian, K.-R. Cho, O. Kavehei, S.-K. Kang, D. Abbott, and S.-M. S. Kang. Memristor MOS content addressable memory (MCAM): Hybrid architecture for future high performance search engines. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(8):1407 –1417, Aug. 2011.

[9] R. Foss and A. Roth. Priority encoder circuit and method for content addressable memory. Technical Report Canadian Patent 2,365, 891, MOSAID Technologies Inc., Apr. 2003.

[10] A. Goel and P. Gupta. Small subset queries and bloom filters using ternary associative memories, with applications. In *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '10, pages 143–154, New York, NY, USA, 2010. ACM.

[11] M. Gokhale, B. Holmes, and K. Iobst. Processing in memory: the terasys massively parallel PIM array. *Computer*, 28(4):23 –31, Apr. 1995.

[12] Q. Guo, X. Guo, Y. Bai, and E. İpek. A resistive TCAM accelerator for data-intensive computing. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44 '11, pages 339–350, New York, NY, USA, 2011. ACM.

[13] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop*, Washington, DC, USA, 2001.

[14] A. Hashmi and M. Lipasti. Accelerating search and recognition with a TCAM functional unit. In *Computer Design, 2008. IEEE International Conference on*, Oct. 2008.

[15] J. L. Henning. SPEC CPU2000: Measuring CPU performance in the new millennium. *IEEE Computer*, 33(7):28–35, July 2000.

[16] Y. Huai. Spin-transfer torque MRAM (STT-MRAM) challenges and prospects. *AAPPS Bulletin*, 18(6):33–40, Dec. 2008.

[17] Intel Corporation. *IA-32 Intel Architecture Optimization Reference Manual*, 2003.

[18] ITRS. *International Technology Roadmap for Semiconductors: 2010 Update*. http://www.itrs.net/links/2010itrs/home2010.htm.

[19] M. Joshi, G. Karypis, and V. Kumar. ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets. In *IPPS*, 1998.

[20] Kawahara, T. and Takemura, R. and Miura, K. and Hayakawa, J. and Ikeda, S. and Young Min Lee and Sasaki, R. and Goto, Y. and Ito, K. and MEGURO, T. and Matsukura, F. and Takahashi, Hiromasa and Matsuoka, Hideyuki and OHNO, H. 2 Mb SPRAM (spin-transfer torque RAM) with bit-by-bit bi-directional current write and parallelizing-direction current read. *IEEE Journal of Solid-State Circuits*, 43(1):109–120, Jan. 2008.

[21] O. D. Kretser and A. Moffat. Needles and haystacks: A search engine for personal information collections. In *Australasian Computer Science Conference*, 2000.

[22] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary. Algorithms for advanced packet classification with ternary CAMs. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '05, pages 193–204, New York, NY, USA, 2005. ACM.

[23] 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 *International Symposium on Computer Architecture*, 2009.

[24] L.-Y. Liu, J.-F. Wang, R.-J. Wang, and J.-Y. Lee.

CAM-based VLSI architectures for dynamic Huffman coding. In *Consumer Electronics, 1994. Digest of Technical Papers., IEEE International Conference on*, June 1994.

[25] M. Madec, J. Kammerer, and L. Hebrard. Compact modeling of a magnetic tunnel junction part II: Tunneling current model. *Electron Devices, IEEE Transactions on*, 57(6):1416–1424, 2010.

[26] S. Matsunaga, K. Hiyama, A. Matsumoto, S. Ikeda, H. Hasegawa, K. Miura, J. Hayakawa, T. Endoh, H. Ohno, and T. Hanyu. Standby-power-free compact ternary content-addressable memory cell chip using magnetic tunnel junction devices. *Applied Physics Express*, 2(2):023004, 2009.

[27] S. Matsunaga, A. Katsumata, M. Natsui, S. Fukami, T. Endoh, H. OHNO, and T. Hanyu. Fully parallel 6T-2MTJ nonvolatile TCAM with single-transistor-based self match-line discharge control. In *VLSI Circuits (VLSIC), 2011 Symposium on*, June 2011.

[28] A. J. Mcauley and P. Francis. Fast routing table lookup using CAMs. In *IEEE INFOCOM*, pages 1382–1391, 1993.

[29] D. McGrath. Everspin samples 64Mb spin-torque MRAM. EETimes, Nov. 2012. http://www.eetimes.com/design/memory-design/4401052/Everspin-samples-64-Mb-spin-torque-MRAM?pageNumber=0.

[30] M. Meribout, T. Ogura, and M. Nakanishi. On using the CAM concept for parametric curve extraction. *Image Processing, IEEE Transactions on*, 9(12):2126 – 2130, Dec. 2000.

[31] Micron Technology, Inc., MT41J128M8. *1Gb DDR3 SDRAM*, 2006.

[32] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary. MineBench: A benchmark suite for data mining workloads. In *Workload Characterization, 2006 IEEE International Symposium on*, Oct. 2006.

[33] M. Oskin, F. Chong, and T. Sherwood. Active pages: a computation model for intelligent memory. In *Computer Architecture, 1998. Proceedings. The 25th Annual International Symposium on*, 1998.

[34] S. Panchanathan and M. Goldberg. A content-addressable memory architecture for image coding using vector quantization. *Signal Processing, IEEE Transactions on*, 39(9):2066 –2078, Sept. 1991.

[35] S. S. P. Parkin, C. Kaiser, A. Panchula, P. M. Rice, B. Hughes, M. Samant, and S. H. Yang. Giant tunnelling magnetoresistance at room temperature with MgO (100) tunnel barriers. *Nature Materials*, 3(12):862–867, 2004.

[36] T.-B. Pei and C. Zukowski. VLSI implementation of routing tables: tries and CAMs. In *INFOCOM '91. Proceedings. Tenth Annual Joint Conference of the IEEE Computer and Communications Societies.*, Apr. 1991.

[37] J. Pisharath, Y. Liu, W. Liao, A. Choudhary, G. Memik, and J. Parhi. NU-MineBench 2.0. Technical report, Northwestern University, August 2005. Tech. Rep. CUCIS-2005-08-01.

[38] J. Potter, J. Baker, S. Scott, A. Bansal, C. Leangsuksun, and R. Asthagiri. ASC: An associative computing paradigm. *Special Issue on Associative Processing, IEEE Computer*, 1994.

[39] B. Rajendran, R. Cheek, L. Lastras, M. Franceschini, M. Breitwisch, A. Schrott, J. Li, R. Montoye, L. Chang, and C. Lam. Demonstration of CAM and TCAM using phase change devices. In *Memory Workshop (IMW), 2011 3rd IEEE International*, May 2011.

[40] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *Proceedings of the 13th International Symposium on High-Performance Computer Architecture*, 2007.

[41] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos. SESC simulator, Jan. 2005. http://sesc.sourceforge.net.

[42] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. In *Proceedings of the 27th annual international symposium on Computer architecture*, ISCA-27, 2000.

[43] S. Sharma and R. Panigrahy. Sorting and searching using ternary CAMs. In *High Performance Interconnects, 2002. Proceedings. 10th Symposium on*, 2002.

[44] R. Shinde, A. Goel, P. Gupta, and D. Dutta. Similarity search and locality sensitive hashing using ternary content addressable memories. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, 2010.

[45] K. Tsuchida, T. Inaba, K. Fujita, Y. Ueda, T. Shimizu, Y. Asao, T. Kajiyama, M. Iwayama, K. Sugiura, S. Ikegawa, T. Kishi, T. Kai, M. Amano, N. Shimomura, H. Yoda, and Y. Watanabe. A 64Mb MRAM with clamped-reference and adequate-reference schemes. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 258 –259, Feb. 2010.

[46] W. Xu, T. Zhang, and Y. Chen. Design of spin-torque transfer magnetoresistive RAM and CAM/TCAM with high sensing and search speed. *IEEE Transactions on Very Large Scale Integration Systems*, 18(1):66–74, Jan 2010.

[47] Z. Zhang, Z. Zhu, and X. Zhang. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *MICRO-33*, 2000.

[48] W. Zhao and Y. Cao. New generation of predictive technology model for sub-45nm design exploration. In *International Symposium on Quality Electronic Design*, 2006. http://ptm.asu.edu/.

[49] J.-G. Zhu. Magnetoresistive random access memory: The path to competitiveness and scalability. *Proceedings of the IEEE*, 96(11):1786 –1798, Nov. 2008.