R-Cache: A Highly Set-Associative In-Package Cache using Memristive Arrays

Payman Behnam, Arjun Pal Chowdhury, and Mahdi Nazm Bojnordi
School of Computing, University of Utah
Salt Lake City, UT, USA
{behnam, arjunmax, bojnordi}@cs.utah.edu

***Abstract*—Over the past decade, three-dimensional die stacking technology has been considered for building large-scale in-package memory systems. In particular, in-package DRAM cache has been considered as a promising solution for high bandwidth and large-scale cache architectures. There are, however, significant challenges such as limited energy efficiency, costly tag management, and physical limitations for scalability that need to be effectively addressed before one can adopt in-package caches in the real-world applications.**

This paper proposes R-Cache, an in-package cache made by 3D die stacking of memristive memory arrays to alleviate the above mentioned challenges. Our simulation results on a set of memory intensive parallel applications indicate that R-Cache outperforms the state-of-the-art proposals for in-package caches. R-Cache improves performance by 38% and 27% over the state-of-the-art direct mapped and set associative cache architectures, respectively. Moreover, R-Cache results in averages of 40% and 27% energy reductions as compared to the direct mapped and set-associative cache systems.

I. Introduction

The ever growing use of data intensive applications across various fields of science and engineering have made large memories and cache architectures inevitable components of future exascale computing platforms. While on-chip memories are limited by power and area consumption, off-chip memory systems suffer from the so called *bandwidth wall* due to pin count limitations [1] and expensive off-chip data movement. Recently, three dimensional (3D) die stacking has enabled integrating multiple DRAM dice within the same processor package that can reach memory bandwidths in excess of Tera bytes per second (TBps) using the through-silicon via (TSV) technologies. High bandwidth memory (HBM), hybrid memory cube (HMC), and wide IO (WIO) are examples of in-package memory systems mainly designed for DRAM [2]–[4]. These recent achievements in memory technologies have motivated numerous researchers to design giga-scale DRAM-based in-package cache architectures.

Despite the high bandwidth interfacing technology, 3D die stacked DRAM access is almost identical to the conventional DDRx memories [5]–[12]. Moreover, current proposals for in-package DRAM provide limited capacity. This limitation may become an even more serious concern for the future data-intensive applications as the DRAM designers cannot provide a clear road map on how the technology will continue to scale [1]. Furthermore, the DRAM power consumption—due to periodic refresh, precharge, static power in large row buffers, and dynamic power for row activation, and data movement over large wires—has resulted in serious thermal issues in 3D die-stacked memories [13]. Apart from these problems, DRAM caches need to manage a large amount of meta data for dirty, valid, tag, and error correction code (ECC) bits that have become a major problem for bandwidth and energy efficiency [5]–[12].

As a result, DRAM alternative technologies have been considered for in-package memories, such as recent proposals that examine 3D memory systems with resistive RAM (RRAM) [14]–[16]. Among all DRAM alternatives, RRAM is a non-volatile memory technology based on memristive devices that exhibit FLASH-like density and fast read speeds comparable to DRAM [17]. Similar to the 3D die-stacked DRAM, building an in-package cache with RRAM remains a significant challenge, mainly due to the excessive capacity and bandwidth consumption for managing meta data. This paper examines R-Cache, an in-package die-stacked memory architecture specifically designed for highly set-associative caches using memristive arrays. The proposed architecture enables energy-efficient and fast in-package caching through the following contributions.

- R-Cache provides a 3D memory structure specifically designed for dense and highly set associative caches. The proposed architecture benefits from the search capabilities in the resistive memory arrays to realize parallel searches for highly set-associative cache lookup. To the best of our knowledge this is the first proposal that specifically designs the memory layers and re-purposes the interface for building an RRAM-based HBM cache.
- Through in-memory tag checking and block management, the proposed R-Cache architecture provides a superior bandwidth efficiency and performance compared to the state-of-the-are direct mapped and set associative cache architectures. Moreover, RRAM eliminates the need for activation and refresh energy, which results in reducing the execution time, energy consumption, and bandwidth utilization of the in-package cache and off-chip memory.

Our simulation results on a set of 12 memory intensive applications indicate that R-Cache improves performance by 38% and 27% over the state-of-the-art direct mapped and

This work was supported in part by the National Science Foundation (NSF) under Grant CCF-1755874 and in part by the University of Utah Seed.

set associative cache architectures, respectively. Moreover, R-Cache results in 40% and 27% reductions the average energy consumption compared to the direct mapped and set associative cache systems.

II. Background and Motivation

This section provides the necessary background knowledge on high bandwidth memory, existing DRAM-based cache architectures, and resistive memory technologies.

A. High Bandwidth Memory

Figure 1 illustrates an example high bandwidth memory (HBM) system including eight 3D stacked DRAM dice and a processor die placed in the same package. The bottom DRAM layer and the processor die are connected to the silicon interposer through micro bumps [18]. Through-silicon vias (TSVs) are used to connect the DRAM layers, which are typically divided into 4-8 vertically independent *vaults* (*channels*) [19]. Each vault is horizontally divided into 8-16 banks that results in up to 128 banks to exploit data-level parallelism in user applications. The banks comprise a hierarchy of sub-banks and sub-arrays with a global row buffer. The DRAM technology necessitates a sequence of `precharge`, `activate`, `read`, and `write` operations prior to transferring data for every memory request. Typically, one HBM controller per vault is needed to orchestrate data movement between the processor and DRAM layers, which involves (1) issuing the right set of commands for each memory request and (2) enforcing the necessary timing constraints among all the generated commands. This paper proposes a high bandwidth RRAM interface realized through the conventional HBM commands with a new set of timing parameters specifically computed for R-Cache operations.

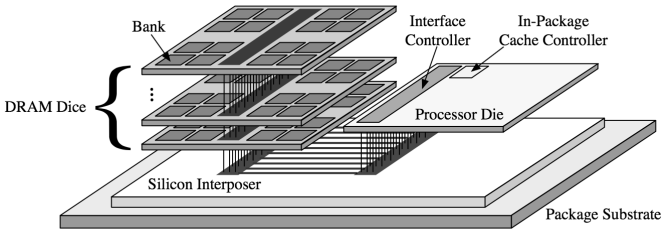

Fig. 1. Illustrative example of a DRAM-base cache architecture using the high bandwidth memory interface.

B. HBM Cache Architectures

Most existing proposals for gigascale cache architectures focus on developing control mechanisms to alleviate the bandwidth and storage overheads of tag checking. Block granularity DRAM caches have millions of cache blocks each of which associated with a tag to make cache lookup possible. Numerous architectures, such as Unison [7], TDC [9], FTDC [10], and Banshee [11], address this problem by increasing the access granularity from less than hundred bytes to multi-kilobyte pages at the cost of more energy and bandwidth consumption per cache fill and eviction. Block level DRAM caches, however, suffer from the high cost of tag storage and management [5], [6], [8], [12]. This paper leverages the higher density and in-memory processing capabilities of the RRAM technology to build a highly set-associative fine-grained gigascale cache architecture for data intensive applications. Figure 2 illustrates the fundamental differences between R-Cache and the existing proposals for direct mapped and set-associative cache architectures.

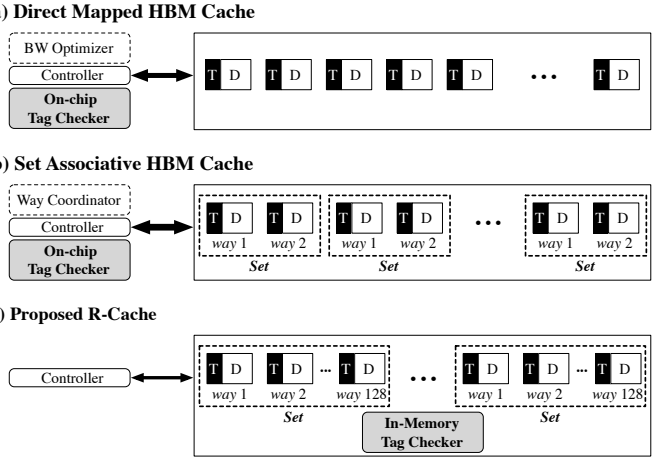

Fig. 2. Fundamental differences between the existing direct mapped (a), set associative (b), and the proposed R-Cache (c) architectures.

Figure 2(a) shows the existing direct mapped HBM based cache architectures that store the tag and data bits in HBM, while the tag checking is carried out on the processor die. Loh and Hill propose a DRAM based architecture that accesses multiple tags or a data block during every HBM access [5]. Instead, Alloy [6] proposes to read both the tag and data (TAD) bits of every cache block within an HBM access [6]. Therefore, the Alloy cache ameliorates latency and bandwidth at the cost of reducing hit rate. Bear [8] proposes to further optimize the HBM bandwidth by reducing the amount of unnecessary traffic between the processor, cache, and main memory.

To improve the limited hit-rate in direct mapped caches, Accord [12] has been recently proposed to develop a set-associative HBM cache. Figure 2(b) shows the key components of such DRAM based cache architecture. Similarly to the direct mapped cache, tag checking is carried out on the processor die; however, the block address mapping is modified to form cache sets in HBM. Such block organizations, on the one hand, increases the cache hit rate; on the other hand, it exerts significant bandwidth overheads for checking all of the tags within every cache set. The Accord cache alleviates this problem by (1) limiting the number of cache ways and (2) employing a way coordinator [20] that reduces the number of unnecessary tag checks per set.

Figure 2(c) shows the proposed R-Cache architecture that addresses both the bandwidth and latency problems through architecting the memory layers with RRAM arrays. R-Cache introduces an energy-efficient in-memory tag checking mechanism that eliminates unnecessary HBM accesses for transferring tag bits between HBM and the processor. Moreover, the novel architecture of R-Cache provides a high set associativity that significantly improves the cache hit rate. Nevertheless,

the higher density of RRAM compared to DRAM holds the promise to keep developing more energy-efficient memory systems for the future data intensive computing.

C. RRAM Technology

Resistive memories are non-volatile, free of leakage power, and immune to radiation induced transient faults. RRAM is one of the most promising memristive devices under commercial development that exhibits excellent scalability, high-speed switching, a high dynamic resistance range, and low power consumption [17]. Numerous array topologies have been proposed in the literature that optimize RRAM for better reliability, density, and computational capabilities [21], [22]. Figure 3 shows example schematic and device structure of the conventional one-transistor, one-memristor (1T-1R) memory cell. The read and write operations are performed by activating the access device through a $wordline$ and applying the required read or write voltage across a $bitline$ and a $bitline^*$ wires.

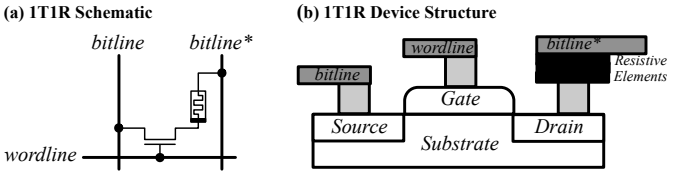

Fig. 3. Example circuit schematic (a) and physical structure (b) of a 1T-1R RRAM cell.

III. R-Cache Architecture

Figure 4 illustrates an example computer system comprising a multicore processor interfaced with three levels of on-chip cache, an in-package 3D die-stacked R-Cache, and an off-chip DRAM memory. Each core has private L1 and L2 cache units. All of the L2 caches are connected to a shared L3 cache. An R-Cache controller connects the L3 cache, the R-Cache layers, and the main memory controller. R-Cache employs a WideIO interface to realize a fine granularity high-bandwidth memory cache. As compared to the conventional DRAM based HBM cache, R-Cache employs RRAM based arrays that provide a higher density as well as additional in-memory search capabilities that enable highly-associative lookup within memory layers. The R-Cache controller and the RRAM arrays are designed to efficiently perform highly set-associative cache lookups under the WideIO interface.

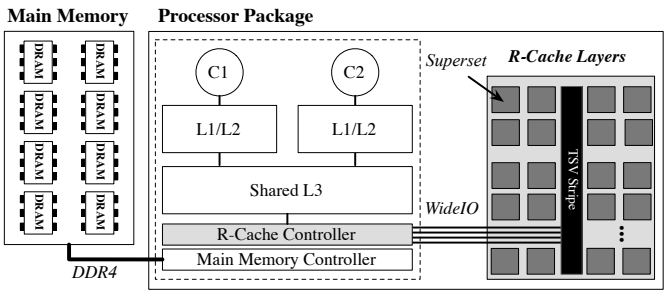

Fig. 4. Illustrative top view of the proposed R-Cache architecture used in a multicore system.

A. R-Cache Memory Organization

R-Cache memory layers are divided into multiple channels; where, each channel comprises independent supersets distributed across eight memory dice. An entire superset is responsible for an ongoing R-Cache memory access. Figure 5 depicts the overall organization of an R-Cache superset divided into multiple sets. Each set consists of a resistive content addressable memory (RCAM) array for storing tags and multiple resistive random accessible memory (RRAM) arrays for maintaining data blocks.

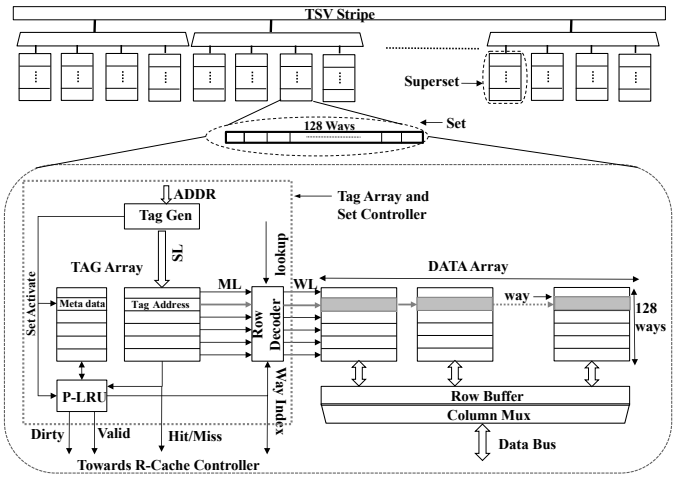

Fig. 5. Illustrative example of the R-Cache set and superset organization.

A search line (SL) signal is generated for every incoming memory address (ADDR) and is used to lookup the RCAM based TAG array. Every 64B data block is distributed among eight RRAM arrays, where every array stores 8B of data. Every set is equipped with a simple in-memory logic, called set controller, that includes a pseudo least-recently used (P-LRU) unit, dirty and invalid bit detectors. P-LRU identifies the least recently used block of the set for eviction prior to every block replacement. During a read/write hit, the row decoder receives a signal for the matching way through match lines (MLs). On a miss, the P-LRU provides an index to the row decoder if an invalid or non-dirty block is found in the set. The row decoder's output is used to activate a word line (WL) of the data arrays. In addition to the address, command, and data buses, R-Cache uses eight output wires from existing WideIO wires for sending the hit/miss, dirty, valid, and LRU index to the R-Cache controller. For example, the data bus invert (DBI) wires may be used for transferring the flag bits from RRAM sets to the R-Cache controller. Unlike the existing solutions for DRAM based in-package cache architectures [9], [10], the highly associative R-Cache design performs all of the tag checks, replacement decision, and dirty block evacuation in memory.

B. Data and Tag Arrays

As shown in Figure 6(a), 2T-2R RRAM cells are employed to build content addressable memory (CAM) arrays for maintaining tags at every cache set. Every cell comprises a pair of 1T-1R memory cells, each of which can be written

individually. The cell can be used for performing bitwise XOR based comparison between a search key and stored tags. The differential format (R and $\overline{R}$) of a tag-bit is stored in every cell. To perform a tag search, a bitwise format of the key value, an incoming tag including the true and complement of each bit, is applied to the $search$ and $\overline{search}$ wires. Prior to the tag search, the *match line* wires are precharged to a high voltage level. Each match line stays at a high level only if all of its tag bits match with the key bits; otherwise, the match line reaches a low voltage level. All of the match lines follow the same process during a tag search to produce an output vector indicating the matches and mismatches in the tag array. (The R-Cache set controller ensures no repeated tags are stored in the tag array; therefore, no more than one match is possible per every tag search.)

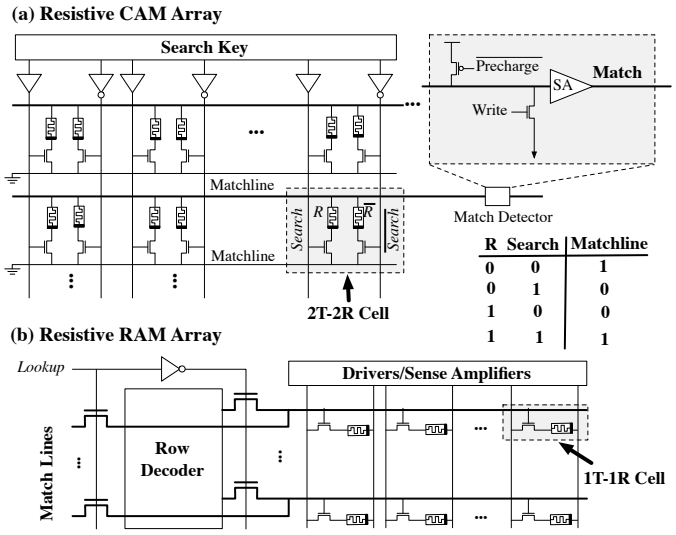

Fig. 6. Illustrative examples of the resistive arrays used for the R-Cache CAM (a) and RAM (b) units.

As shown in Figure 6(b), the data arrays are designed based on the conventional 1T-1R RRAM cells. Every memory cell comprises a transistor and a memristor to store a bit of information. A single row of the sub-array is activated on every access using the signals generated by the row decoder or the CAM match lines. A control signal, called *lookup*, is used to choose between a normal read/write operation and a cache lookup.

C. R-Cache Controller

Figure 7 shows the R-Cache controller as a gateway between L3, 3D R-Cache layers, and the main memory controller. First, an input stream of read and write requests from L3 is routed to the R-Cache controller through a channel router. The requests are then converted to the HBM commands. Ultimately, data is exchanged between the R-Cache controller and the memory layers through three main components: a *Transaction Unit*, a *Transaction Queue*, and a *Data Buffer*.

1) Transaction Unit: The transaction unit (TU) is responsible for converting the memory requests to transactions, and initiating block eviction and fill operations based on cache miss and dirty indication. In addition, all of the timing

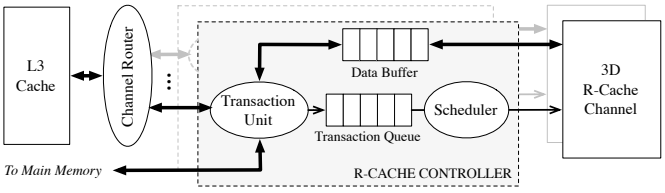

Fig. 7. An overview of the R-Cache controller .

requirements dictated by the WideIO interface are ensured by TU. The cache block hit/miss and dirty evaluation, however, are carried out inside the memory dice and are signaled to the controller through the WideIO interface.

2) Transaction Queue: A transaction queue is employed by every channel controller that stores up to 64 R-Cache transactions. Each entry of the queue is dequeued and translated to a set of WideIO commands (e.g., `read`, `write`, `precharge`, and `activate`) using a *scheduler* to manage the memory layers and orchestrate data on the R-Cache interface.

R-Cache set the data burst and activation size to a single cache block to limit the energy consumption; therefore, every R-Cache transaction results in activating 64B data blocks.[1]

3) Data Buffer: The R-Cache controller employs a data buffer for storing and transferring data between the L3 cache and the 3D memory layers.

D. R-Cache Timing Protocol

Similar to prior work on non-volatile main memories [23], R-Cache employs the WideIO interface, originally designed for DRAM arrays, with proper timing parameters to operate the resistive memory layers with longer write latencies. The important changes in the timing parameters are related to the row precharge (tRP) and the read to precharge (tRTP) delays, which are set to zero; all other timing parameters are computed based on the RRAM read, write, and search latencies. The details of parameters are provided in Table III.

IV. R-Cache Control Flows

This section explains the proposed control flows required for serving every memory request arriving at R-Cache. The control tasks for every new request are divided between the controller and an R-Cache set. Two tightly coordinated control flows are proposed such that one is responsible for transaction processing on the processor die and the other one is performed by the set controller to maintain the set status.

A. Transaction Processing

The transaction unit inside the R-Cache controller is responsible for generating appropriate transactions for the read and write requests generated by L3. Table I shows the transactions supported by the proposed transaction unit. The R-Cache *read* and *write* require a tag check followed by a data access at the target set. In contrast, the *read_dirty* and *write_fill* transactions are issued under certain conditions to perform pure read and

[1]Our experiments indicate that increasing the number of bits per activated rows in 3D RRAM cache results in negligible performance improvements but significant energy consumption.

write accesses with no tag check. These transactions carry the necessary indexing information for cache block replacement under write misses. During these transactions the *lookup* signal is set to low for the resistive RAM arrays (see part (b) of Figure 6). The *read_dirty* transaction is used to read an identified dirty block from the R-Cache layers before a write update; whereas, the *write_fill* transaction updates the contents of a requested block after eviction from the R-Cache set.

TABLE I
R-CACHE TRANSACTIONS.

Name	Tag Check	Description
read	Required	Read requests from the L3 cache are issued to R-Cache as normal read transaction.
write	Required	1- In the case of an L3 block eviction, write backs from L3 are placed as ordinary writes in the transaction queue to write into R-Cache; 2-In the case of R-Cache read miss, read fills from main memory are converted to normal writes.
read_dirty	Not Required	This transaction is issued when a write miss happens and a dirty block is found for replacement.
write_fill	Not Required	This transaction is issued after a write miss once a dirty block is evicted and the target block is read from the main memory.

Please notice in the case of a read miss, a *read_fill* transaction may not be sufficient to place an incoming block from main memory to R-Cache. The main reason is the status of the set that may change during the long latency between detecting a cache miss and receiving the missing block from main memory. Therefore, a *write* transaction comprising a fresh tag checking operation followed by updating the cache contents is necessary. In contrast, a write miss carries data with the request from L3 and can immediately update the contents of the memory layers without any tag checking.

B. Set Maintenance Processing

R-Cache relies on a distributed control mechanism between the memory arrays and the R-Cache controller for set maintenance that includes in-memory tag checking, block eviction, tracking the replacement candidates, and block transfer. Figure 8 shows the flow of operations required for set maintenance upon receiving a command at the memory arrays which happens inside HBM. R-Cache follows two phases for processing every memory request. The first phase comprises a normal read/write operation with flag outputs indicating hit/miss and whether a $read\_dirty$ or $writ\_fill$ transaction is required (Figure 8). In the latter case, the second phase (Figure 8) will be performed as the set controller receives the signals from the target set. Here, we explain the required operations for serving read and write requests, separately.

1) Read Operation: The target R-Cache set receives a `read` command over the WideIO interface. In the case of a *read_dirty*, no lookup operation is required and the way index provided along the transaction is used to directly access the resistive RAM arrays. Otherwise, a normal *read* transaction requires a tag check to find a matching way prior to accessing the data blocks. Along with the read data, the R-Cache set returns status bits including **hit** and **dirty** flags. A response is then sent to the R-Cache controller based on the current state of the set. Figure 9 illustrates the flow of operations performed

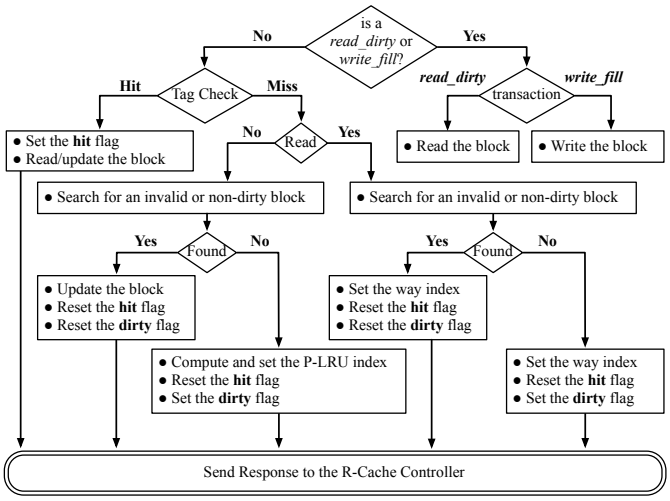

Fig. 8. Flow of operations required for set maintenance in memory layers.

by the R-Cache controller based on the set response. On a cache hit, the read block is sent to the L3 cache when the response arrives at the R-Cache controller. If the response indicates a read miss, the R-Cache controller issues a read request to the main memory that fetches the missing block. Once the data is ready on chip, (1) the block is sent to the L3 cache and (2) a write transaction is generated and enqueued in the transaction queue for updating the R-Cache memory arrays.

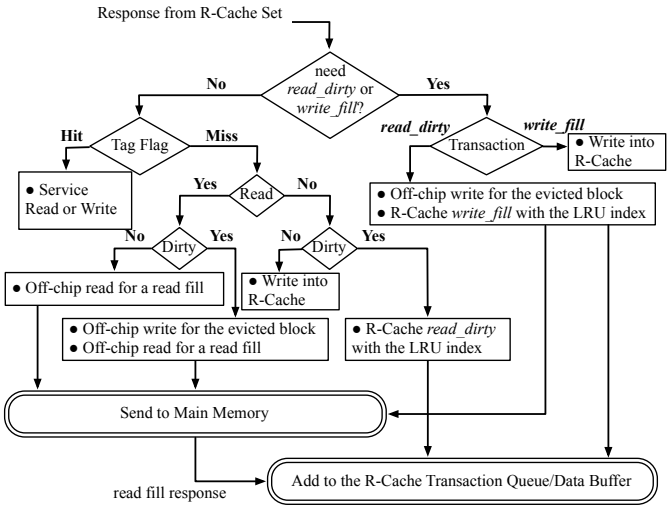

Fig. 9. Flow of the control operations in the R-Cache controller in response to an R-Cache access.

2) Write Operation: Similar to the read operation, write transactions are generated and sent by the R-Cache controller to the memory layers (Figure 9). A *write_fill* transaction directly updates a particular way of the target set using the way index provided along the transaction. However, an in-memory tag check is required for normal *write* transactions. On a hit or an invalid block being found in the set, the block contents are directly updated by the transferred data. Otherwise, the in-memory P-LRU logic computes the way index to be replaced by the incoming block. The R-Cache set returns the selected

index and the **hit** and **dirty** flags to the R-Cache controller for eviction and replacement.

Overall, a **hit** or a non-**dirty** signal to the controller indicates a successful cache write operation. On a **miss** and a **dirty** signal with an LRU index for block eviction, the controller generates and sends a *read_dirty* transaction to the transaction queue. Then, the set directly reads the data from the provided index and returns it to the controller. Upon receiving the evicted data from the R-Cache set, the R-Cache controller issues a write request to main memory for the evicted block. At the same time, it generates a *write_fill* transaction to update the contents of the set. Upon receiving the *write_fill* transaction at the set, the contents of memory arrays are updated.

V. Experimental Setup

In this section, the simulation environment, methodology of experiments, characteristics of applications, and power and performance evaluation tools are explained.

A. Methodology

We heavily modify CACTI 7.0 [24] and borrow some components from NVSIM [25] and NVSIM-CAM [26] to develop an in-house tool, named NVCACTI, to model the proposed R-Cache architecture for area, power, energy, and latency estimations. The newly developed NVCACTI supports non-volatile memories with the multi-banking capabilities and parameterized set and superset modeling. (Notice that NVSim and NVSim-CAM do not support multi banking; whereas, CACTI does not support non-volatile memories.)

To carry out cycle-accurate performance and energy simulations, we develop an HBM simulator interfaced with a multicore processor and DDR4 main memory by heavily modifying the ESESC [27] simulator. McPAT [28] is used in coordination with ESESC to estimate the processor power and system energy. For all of the circuit modeling efforts, we employ the 22nm CMOS technology and the RRAM parameters from prior work [29]. Table II shows the RRAM parameters used for modeling R-Cache memory layers.

TABLE II
The RRAM parameters used for modeling R-Cache arrays.

Rhigh/Rlow	Peak Voltage	Peak Current	Speed	Retention	Endurance
>10	<2V	170μA	10 ns	10 years@85°C	10^9

B. System Configuration

Table III shows the simulation parameters for three different HBM structures including DRAM-Cache, RRAM-Cache, and R-Cache. The memory capacity and the necessary timing parameters related to the HBM interface for the RRAM and R-Cache systems are extracted using NVCACTI. We consider iso area configurations for all of the evaluated memory systems. We use CACTI to estimate the die area consumption of an eight-layer, 1GB 3D stacked DRAM. For the RRAM baselines and R-Cache, the NVCACTI tool is employed to find optimized configurations based on energy-delay products. All of the evaluated designs are limited to fit within the same area as the one of the DRAM cache. (As the RRAM technology provides denser memory arrays than DRAM, the RRAM and R-Cache systems can store up to 2.25GB data within the same area size of a 1GB DRAM.) The HBM timing parameters are then set accordingly.

TABLE III
The evaluated system configurations.

Processor			
Core	eight 4-issue OoO cores, 128 ROB entries, 3.2 GHz		
IL1/DL1 cache	32KB, 2-way, LRU, 64B block		
L2 cache	128KB, 4-way, LRU, 64B block		
L3 cache	1MB, 8-way, LRU, 64B block		
HBM Cache			
	DRAM	**RRAM**	**R-Cache**
Specifications	1GB, 8 channels, 16 banks/channel, 800MHz DDR4	2.25GB, 8 channels, 64 banks/channel, 800MHz DDR4	2.25GB, 8 channels, 64 supersets/channel, 800MHz DDR4
Timing (CPU cycles)	tRCD: 44, tCAS: 44, tRP: 44, tRTP: 46, tRAS: 112, tWR: 4	tRCD: 11, tCAS: 0, tRP: 0, tRTP: 0, tRAS: 44, tWR: 205	tRCD: 15, tCAS: 0, tRP: 0, tRTP: 0, tRAS: 44, tWR: 207
Off-Chip Main Memory			
Specifications	32GB, DDR4, 1 channel, 2 ranks/channel, 8 banks/rank		
Timing (CPU cycles)	tRCD: 44, tCCD: 61, tWTR: 31, tWR: 4, tRTP: 46, tRP: 44, tRRD: 16, tRAS: 112, tRC: 271, tFAW: 181		

To assess the energy and performance potentials of the proposed R-Cache system, we evaluate four different architectures: a DRAM Alloy cache, an RRAM Alloy cache, a 2-way RRAM Accord cache, and R-Cache. Both Alloy and Accord are the state-of-the-art architectures for direct mapped and set-associative DRAM cache systems. To create strong baselines for comparisons, we develop versions of the two architectures that significantly benefit from the energy-efficiency and density of the RRAM technology. All of the evaluated cache systems rely on the block granularity data and tag storage; where, every 64B data is associated with a 8B tag. Thanks to the efficient 2T-2R RCAM arrays, each R-Cache set realizes 128 ways; whereas, the Accord cache can efficiently implements a 2-way set associative cache. Notice that in Accord increasing the number of associative ways per set results in a significant bandwidth and energy consumption with performance degradation.

C. Benchmark Applications

We choose a mix of 12 data-intensive benchmark applications from three parallel suites, namely Phoenix [30], SPLASH-2 [31], and NAS [32]. Applications are compiled using GCC with the -O3 optimization flag. Table IV shows the benchmarks description and their input sets. All of these parallel applications are simulated to completion for performance evaluations. Similar to prior work [11], we define a fixed capacity quota for all of the evaluated applications to exert enough pressure on the HBM cache interface.

TABLE IV
Applications and data sets.

Label	Benchmarks	Suite	Input
FT	Fourier Transform	NAS	Class A
IS	Integer Sort	NAS	Class A
MG	Multi-Grid	NAS	Class A
CG	Conjugate Gradient	NAS	Class A
CH	Cholesky	SPLASH-2	tk29.0
RDX	Radix	SPLASH-2	2M integer
OCN	Ocean	SPLASH-2	514x514 ocean
FFT	FFT	SPLASH-2	1048576 data points
FMM	Fast Multiple Methods	SPLASH-2	1048576 data points
LU	Lower/Upper Triangular	SPLASH-2	isiz02=64
BRN	Barnes	SPLASH-2	16K particles
HIST	Histogram	Phoenix	100MB file
LREG	Linear Regression	Phoenix	50MB key file

VI. Evaluation

We evaluate the area overheads, system performances, HBM cache hit rates, bandwidth efficiencies, and system energy

consumption of the proposed R-Cache and baseline DRAM and RRAM architectures.

A. Area Consumption

Compared to pure RRAM based cache architectures, the proposed R-Cache replaces some of the RRAM arrays with RCAM to store tags and employs CMOS logic on the memory dice for in-memory tag checking. We observe that the peripheral circuits (e.g., sense amplifiers and line drivers) in RRAM and RCAM arrays significantly contribute to die area consumption; therefore, replacing the 1T-1R RRAM cells with 2T-2R RCAM cells does not introduce significant area overheads. However, the additional logic for in-memory set management requires a considerable amount of area overheads, which results in limiting the number of supersets per channel. Through a thorough design exploration using the newly developed NVCACTI tool, we find an R-Cache configuration with 64 supersets and 8 sets per superset that requires only 6.9% area overhead compared to the optimized RRAM HBM.

B. System Performance

Figure 10 shows the relative system performance of the RRAM Alloy, Accord, and R-Cache architectures while executing all of the 12 benchmark applications. All of the numbers are normalized to the DRAM Alloy baseline architecture. Compared to the DRAM cache, all of the RRAM based caches achieve higher performances; for example, RRAM Alloy achieves an average 2.18× performance improvement due to replacing DRAM technology with RRAM. RRAM Accord improves the system performance by 8% over the RRAM Alloy mainly through increasing the cache hit rate. R-Cache, however, achieves a superior performance compared to all of the baselines; in particular, it gains an average of 38% performance improvement over the RRAM Alloy cache and 27% over the RRAM Accord architecture.

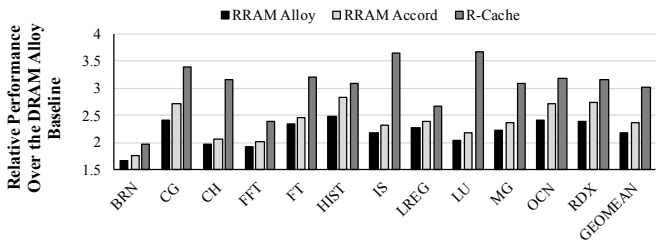

Fig. 10. Relative performance.

C. Cache Hit Rate

The superior performance of R-Cache is due to improving the hit rate and bandwidth efficiency at the same time. Figure 11 illustrates the hit rates of all the evaluated applications on the RRAM based systems using the same capacity. The average hit rate of the direct mapped cache (i.e., RRAM Alloy) is about 67%; the average hit rate of the 2-way RRAM Accord is 70%; while, R-Cache reaches an average hit rate of 90.3%, which is due to its high set-associativity.

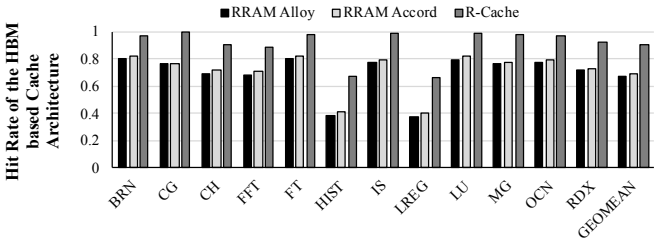

Fig. 11. Observed hit rates for the RRAM based architectures.

D. Bandwidth Efficiency

We compute the relative amount of data and tag bits transferred over the HBM interface per every memory request to evaluate the efficiency of bandwidth utilization in every cache system. Figure 12 shows the average per-access bandwidth utilization (PABU) for all of the 12 applications executed on the evaluated cache systems. All of the numbers are normalized to the DRAM Alloy cache. RRAM Alloy provides the same PABU as the DRAM Alloy cache. RRAM Accord increases PABU by an average of 19% due to the additional bandwidth consumptions for 2-way tag checking. R-Cache, however, reduces PABU by 31% compared to the DRAM Alloy due to eliminating the tag checking on the HBM interface.

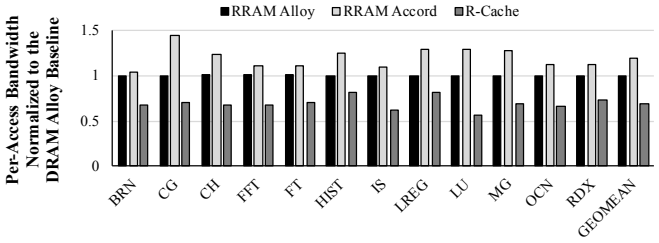

Fig. 12. Average bandwidth utilizations per cache access.

E. System Energy

R-Cache significantly improves the system energy by (1) eliminating the need for refresh and activation, (2) reducing data movement on the HBM interface, and (3) improving the overall execution time. Figure 13 illustrates the system energy consumption for the evaluated cache architectures. The proposed R-Cache architecture achieves averages of 70% and 40% reductions in system energy compared to the DRAM and RRAM Alloy caches, respectively. As compared with RRAM Accord, R-Cache achieves an average of 27% reduction in the overall system energy.

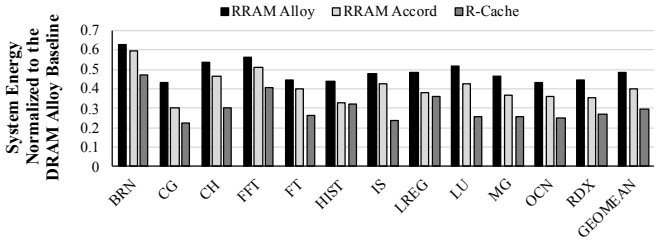

Fig. 13. Normalized system energy consumption.

VII. R-Cache Lifetime

One of the main challenges in designing RRAM-based systems is the limited number of writes can be performed before a memory cell stops functioning. The problem may translate into a short system lifetime. Generally, a longer memory lifetime may be achieved by (1) employing RRAM materials that endure more writes and (2) distributing write operations across all memory location evenly (i.e., wear leveling [33]). Nevertheless, not all applications generate high write rates. For example, Figure 14 shows the average and maximum rates of writes per block generated by the evaluated applications. OCN produces the highest maximum rate among all the benchmarks. Assuming a write endurance of 10^9 [29] and a processor frequency of 3.2GHz, OCN results in 2.89 months of life for the most frequently written block. The memory lifetime, however, may be extended significantly through page-grained wear-leveling: less frequently written blocks be assigned to highly updated pages on every execution. This optimization results in a 400-year lifetime for OCN.

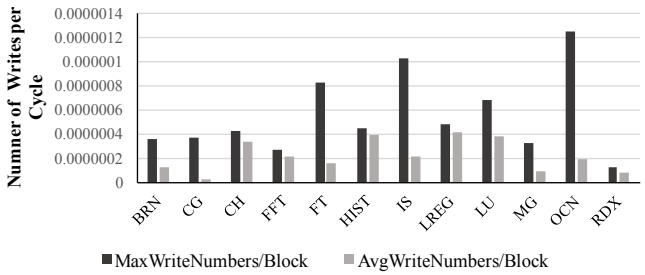

Fig. 14. The average and maximum rates of writes per R-Cache blocks.

VIII. Conclusions

This paper examined a novel in-package cache architecture, called R-Cache. The proposed cache system leverages the RRAM technology to design a highly set-associate cache structure in the processor package using 3D die stacking techniques. R-Cache eliminates the need for on-die tag checking due to integrating logic and set-associative arrays on the memory layers. The superior performance and energy efficiency of R-Cache holds the promise to build efficient computer systems for the future large scale data processing.

References

[1] ITRS, *International Technology Roadmap for Semiconductors: 2013 Edition*, http://www.itrs.net/Links/2013ITRS/Home2013.htm.

[2] H. M. C. Specification, "1.0, accessed date: 08/13/2016."

[3] S. JEDEC, "High bandwidth memory (hbm) dram," *JESD235*, 2013.

[4] H. Micron, "gen2," *Micron*, 2013.

[5] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked dram caches," in *Micro*. ACM/IEEE, 2011, pp. 454–464.

[6] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," in *Micro*. ACM/IEEE, 2012, pp. 235–246.

[7] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in *Micro*. ACM/IEEE, 2014, pp. 25–37.

[8] C. Chou, A. Jaleel, and M. K. Qureshi, "Bear: techniques for mitigating bandwidth bloat in gigascale dram caches," in *ISCA*. IEEE, 2015, pp. 198–210.

[9] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee, "A fully associative, tagless dram cache," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 211–222.

[10] H. Jang, Y. Lee, J. Kim, Y. Kim, J. Kim, J. Jeong, and J. W. Lee, "Efficient footprint caching for tagless dram caches," in *HPCA*. IEEE, 2016, pp. 237–248.

[11] X. Yu, C. J. Hughes, N. Satish, O. Mutlu, and S. Devadas, "Banshee: Bandwidth-efficient dram caching via software/hardware cooperation," in *Micro*. IEEE/ACM, 2017, pp. 1–14.

[12] V. Young, C. Chou, A. Jaleel, and M. Qureshi, "Accord: Enabling associativity for gigascale dram caches by coordinating way-install and way-prediction," in *ISCA*. IEEE, 2018.

[13] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso *et al.*, "Die stacking (3d) microarchitecture," in *Micro*. ACM/EEE, 2006, pp. 469–479.

[14] D. L. Lewis and H.-H. S. Lee, "Architectural evaluation of 3d stacked rram caches," in *3DIC*, 2009.

[15] Y. Deng, H.-Y. Chen, B. Gao, S. Yu, S.-C. Wu, L. Zhao, B. Chen, Z. Jiang, X. Liu, T.-H. Hou *et al.*, "Design and optimization methodology for 3d rram arrays," in *IEDM*. IEEE, 2013, pp. 25–7.

[16] M. Yu, Y. Cai, Z. Wang, Y. Fang, Y. Liu, Z. Yu, Y. Pan, Z. Zhang, J. Tan, X. Yang *et al.*, "Novel vertical 3d structure of tao x-based rram with self-localized switching region by sidewall electrode oxidation," *Scientific reports*, vol. 6, p. 21020, 2016.

[17] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *nature*, vol. 453, no. 7191, p. 80, 2008.

[18] N. Kim, D. Wu, D. Kim, A. Rahman, and P. Wu, "Interposer design optimization for high frequency signal transmission in passive and active interposer using through silicon via (tsv)," in *ECTC*. IEEE, 2011, pp. 1160–1167.

[19] G. H. Loh, "3d-stacked memory architectures for multi-core processors," in *ISCA*. IEEE, 2008, pp. 453–464.

[20] A. Seznec, "A case for two-way skewed-associative caches," in *SIGARCH computer architecture news*, vol. 21, no. 2. ACM, 1993, pp. 169–178.

[21] D. Wust, M. Biglari, J. Kncodtel, M. Reichenbach, C. Söll, and D. Fey, "Prototyping memristors in digital system with an fpga-based testing environment," in *PATMOS*. IEEE, 2017, pp. 1–7.

[22] Y. K. Rupesh, P. Behnam, G. R. Pandla, M. Miryala, and M. N. Bojnordi, "Accelerating k-medians clustering using a novel 4t-4r rram cell," *TVLSI*, vol. PP, no. 99, pp. 1–14, 2018.

[23] J. Meza, J. Li, and O. Mutlu, "Evaluating row buffer locality in future non-volatile main memories," 2012.

[24] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *TACO*, vol. 14, no. 2, p. 14, 2017.

[25] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *TCAD*, vol. 31, no. 7, pp. 994–1007, July 2012.

[26] S. Li, L. Liu, P. Gu, C. Xu, and Y. Xie, "Nvsim-cam: A circuit-level simulator for emerging nonvolatile memory based content-addressable memory," in *ICCAD*. IEEE, 2016, pp. 1–7.

[27] E. K. Ardestani and J. Renau, "ESESC: A fast multicore simulator using time-based sampling," in *HPCA*. IEEE, 2013, pp. 448–459.

[28] 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 *Micro*. ACM/IEEE, 2009, pp. 469–480.

[29] Z. Wei, Y. Kanzawa, K. Arita, Y. Katoh, K. Kawai, S. Muraoka, S. Mitani, S. Fujii, K. Katayama, M. Iijima *et al.*, "Highly reliable taox reram and direct evidence of redox reaction mechanism," in *Electron Devices Meeting, 2008. IEDM 2008. IEEE International*. IEEE, 2008, pp. 1–4.

[30] R. M. Yoo, A. Romano, and C. Kozyrakis, "Phoenix rebirth: Scalable MapReduce on a large-scale shared-memory system," in *ISWC*, 2009.

[31] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *ISCA, IEEE*, 1995.

[32] D. H. Bailey *et al.*, "NAS parallel benchmarks," NASA Ames Research Center, Tech. Rep., March 1994, tech. Rep. RNR-94-007.

[33] S. E. Wells, "Method for wear leveling in a flash eeprom memory," Aug. 23 1994, uS Patent 5,341,339.