UC Irvine
UC Irvine Previously Published Works

Title
Multicopy Cache: A Highly Energy-Efficient Cache Architecture

Permalink
https://escholarship.org/uc/item/1jt3k8nd

Journal
ACM Transactions on Embedded Computing Systems, 13(5s)

ISSN
1539-9087

Authors
Chakraborty, Arup
Homayoun, Houman
Khajeh, Amin
et al.

Publication Date
2014-12-15

DOI
10.1145/2632162

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Multicopy Cache: A Highly Energy-Efficient Cache Architecture

ARUP CHAKRABORTY, University of California, Irvine
HOUMAN HOMAYOUN, University of California, San Diego
AMIN KHAJEH, Qualcomm Inc.
NIKIL DUTT, AHMED ELTAWIL, and FADI KURDAHI, University of California, Irvine

Caches are known to consume a large part of total microprocessor energy. Traditionally, voltage scaling has been used to reduce both dynamic and leakage power in caches. However, aggressive voltage reduction causes process-variation-induced failures in cache SRAM arrays, thus compromising cache reliability. We present MultiCopy Cache (MC^2), a new cache architecture that achieves significant reduction in energy consumption through aggressive voltage scaling while maintaining high error resilience (reliability) by exploiting multiple copies of each data item in the cache. Unlike many previous approaches, MC^2 does not require any error map characterization and therefore is responsive to changing operating conditions (e.g., Vdd noise, temperature, and leakage) of the cache. MC^2 also incurs significantly lower overheads compared to other ECC-based caches. Our experimental results on embedded benchmarks demonstrate that MC^2 achieves up to 60% reduction in energy and energy-delay product (EDP) with only 3.5% reduction in IPC and no appreciable area overhead.

Categories and Subject Descriptors: B.3.1 [**Semiconductor Memories**]: Static Memory (SRAM); B.3.2 [**Design Styles**]: Cache Memories; B.1.3 [**Control Structure Reliability, Testing and Fault-Tolerance**]: Error Checking, Redundant Design

General Terms: Algorithms, Design, Reliability, Theory

Additional Key Words and Phrases: Variation-aware cache, low-power cache, low-power memory organization, low-power design, fault tolerance

ACM Reference Format:
Arup Chakraborty, Houman Homayoun, Amin Khajeh, Nikil Dutt, Ahmed Eltawil, and Fadi Kurdahi. 2014. Multicopy cache: A highly energy-efficient cache architecture. ACM Trans. Embedd. Comput. Syst. 13, 5s, Article 150 (July 2014), 27 pages.
DOI: http://dx.doi.org/10.1145/2632162

1. INTRODUCTION

As ITRS roadmap predicts [ITRS 2008; Behmann 2009], in the continued pursuit of Moore's law, power densities will continue to affect reliability of both embedded SoCs and high-performance desktop/server processors. Although the logic content and throughput of the systems will continue to increase exponentially, a flat curve must be maintained for dynamic and leakage power in order to prolong battery life, maintain cooling costs, and mitigate the adverse effects of increased power densities on reliability. The resulting *power management gap* must be addressed through various means including architectural techniques. Caches are already known to consume a

---

This is an expanded version of the paper titled "E < MC^2: Less Energy through Multi-Copy Cache" published in CASES 2010.
Authors' addresses: A. Chakraborty (corresponding author), Center for Embedded Computer Systems, University of California, Irvine; email: arup@ics.uci.edu; H. Homayoun, Department of Computer Science and Engineering, University of California, San Diego; A. Khajeh, Qualcomm Inc., Austin, TX; N. Dutt, A. Eltawil, and F. Kurdahi, Center for Embedded Computer Systems, University of California, Irvine.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2014 ACM 1539-9087/2014/07-ART150 $15.00
DOI: http://dx.doi.org/10.1145/2632162

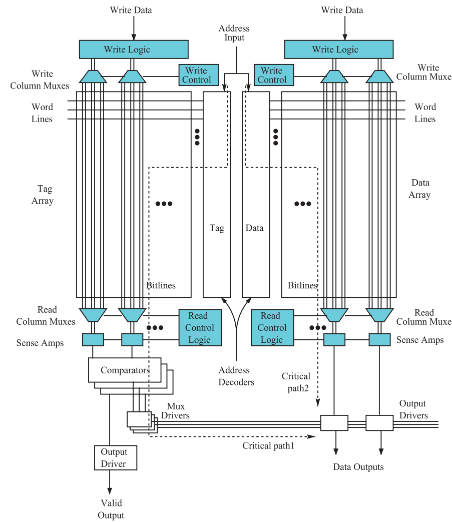


Fig. 1. Typical cache structure (reproduced from Mamidipaka and Dutt [2004]).

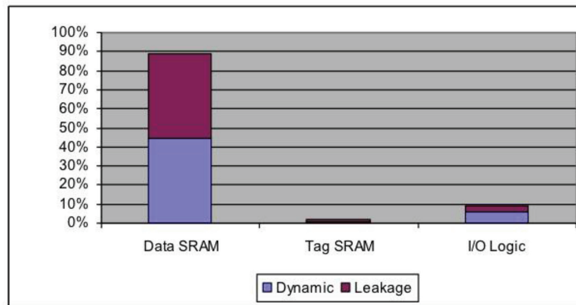


Fig. 2. Power consumption of 16KB cache (using CACTI 4.1 for 70nm technology).

large amount of power, about 30–70% of total processor power [Wong et al. 2007; Zhang et al. 2005] for embedded systems, and on-chip cache size will continue to grow due to device scaling coupled with performance requirements. Therefore, in order to manage total power consumption and reliability of the system, it is important to manage power and reliability of the caches.

A typical cache structure is shown in Figure 1. It consists of mainly three components: (a) an SRAM array for storing data; (b) a much smaller SRAM array to store the tag; and (c) input/output logic, including decoder, comparator, and output muxes and drivers. Figure 2 shows the breakdown of the power consumption in a 16KB cache for 70nm, estimated using CACTI 4.1 [Tarjan et al. 2006]. Clearly the data SRAM consumes about 88% of total power, while the rest is shared by the tag SRAM and input/output logic. Hence, in order to reduce the power consumption of the cache, one must particularly focus on the data SRAM.

Traditionally, voltage scaling has been used to reduce the dynamic and leakage power consumption of the cache. However, aggressive voltage scaling causes process-variation-induced failures in SRAM cells such as read access failures, destructive read failures, and write failures [Mukhopadhyay et al. 2005; Chen et al. 2005]. Since applications may not be tolerant to even a single bit error, caches must be operated at a

high Vdd with a very low probability of failure, leading to high energy consumption. However, by exploiting mechanisms that allow a cache to become inherently resilient to large numbers of cell failures, we can operate the cache at a lower Vdd and thus gain significant energy savings.

In this work, we propose MultiCopy Cache (MC^2), a novel cache architecture that significantly enhances the reliability of the cache by maintaining multiple copies of every data item. Whenever data is accessed, multiple copies of the accessed data are processed to detect and correct errors. Specifically in this work, two copies of each clean data and three copies of each dirty data are maintained in the cache to increase reliability. MC^2 is particularly useful for embedded applications since their working-set sizes are often much smaller than existing cache sizes, and the unused cache space can be effectively used for storing multiple copies, achieving error resiliency through redundancy. Such a cache has high reliability and can be subject to aggressive voltage scaling, resulting in significant reduction in energy consumption. Moreover, since errors are dynamically detected and corrected, MC^2 does not need any a priori error characterization of the cache. Also, compared to other existing cache architectures that exploit redundancy (e.g., ECC), MC^2 incurs minimal performance and area overheads. MC^2 may decrease cache capacity significantly but, as we show later, for embedded applications this results in only modest losses in performance. Our experimental results on embedded benchmarks show that, compared to a conventional cache operating at nominal Vdd, MC^2 reduces energy consumption by up to 60%, with only about 3.5% loss in performance and no appreciable area overhead.

The rest of the article is organized as follows: Section 2 discusses the opportunity for efficiently increasing cache reliability and some background related to SRAM reliability. Section 3 introduces the MC^2 architecture and Section 4 discusses related works. Section 5 presents the hardware implementation and its overheads. Section 6 evaluates the architecture in terms of performance and energy for a set of embedded applications, and Section 7 concludes.

2. BACKGROUND

2.1. Opportunity: Small Working-Set Sizes

We exploit the fact that the working-set sizes of many embedded applications are much lower than available cache space in modern embedded processors. Fritts and Wolf [2000] define working-set size of an application as the cache size in which the miss rate decreases dramatically (at least 50%) with respect to smaller cache size. In absence of such a dramatic decrease, working-set size is defined to be the size that reduces miss rate below 2%. Fritts et al. [1999] showed that for multimedia applications, working-set size for instructions is less than 8KB and that for data is less than 32KB. Guthaus et al. [2001] showed that for most embedded applications, instruction and data working-set size is less than 4–8KB. Our own investigation for the MiBench embedded suite (Figure 3) shows that, although there are a few benchmarks with a working-set size of 16–32K, most of the benchmarks have a working-set size of 8K or less, with about 50% of the applications having less than 2K as working-set size. On the other hand, as Table I shows, modern SoCs and processors typically have L1 cache of size 16–64KB and L2 cache of size up to 2MB, demonstrating a significant portion of the cache (outside of the working set) is not used for many embedded applications.

We exploit this opportunity to utilize the extra cache space to create an efficient error control mechanism *embedded* in the cache by maintaining multiple copies of each data item. A number of techniques have been previously proposed to increase reliability of caches and SRAM memories. Some of these techniques like parity and ECC [Hsiao 1970; ARM 2010; Sohi 1989] have the ability to *dynamically* detect and correct only

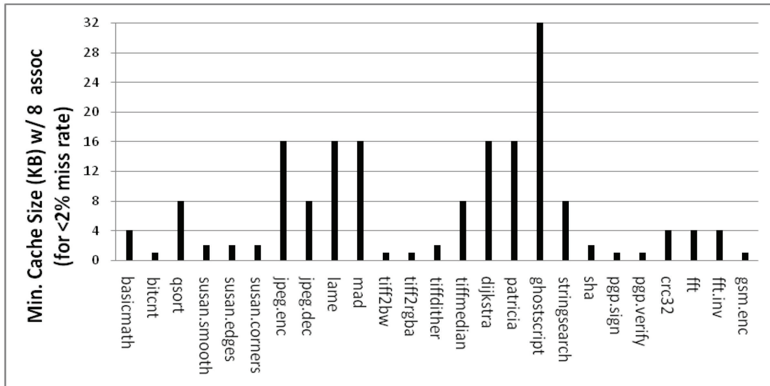


Fig. 3. Minimum cache size with associative 8 needed for <2% miss rate.

Table I. L1 and L2 Cache Sizes for Modern Microprocessors

Processor	Intel Xscale	ARM Cortex A8	ARM Cortex A9	Freescale QorIQ P2
L1 size	32K	16-32K	16-64K	32K
L2 size	512K	2M	2M	512K

a limited number of errors but incur high penalty in access latency and area. Other techniques [Wilkerson et al. 2008; Agarwal et al. 2005; Djahromi et al. 2007; Makhzan et al. 2007; Sasan et al. 2009a, 2009b; Shirvani and McCluskey 1999] provide high error tolerance but require a cache error map of various resolutions (per-byte to per-cache line) that must be generated by BIST whenever there is a change in operating conditions such as Vdd and frequency. Since SRAM failures are highly dynamic and influenced by several conditions beyond the control of users, such failures may not be captured by infrequent BIST characterizations (as explained later). In contrast to previous and related works, our MC^2 architecture:

—can dynamically detect a very high number of errors in SRAM arrays;
—does not require any BIST characterization;
—is responsive to dynamic changes in the SRAM error pattern;
—unlike SECDED, incurs only a minimal impact on both access latency and SRAM area and yet has high error tolerance; and
—enables aggressive Vdd scaling, yielding significant reduction in energy consumption.

2.2. Process Variation and SRAM Reliability

Failures of SRAM cells can be of various types caused by different reasons. For example, there may be manufacturing defects leading to permanent open/short circuits in SRAM cells, causing permanent failures. These are known as hard failures [Mukhopadhyay et al. 2005]. There may also be transient errors, caused by radiation particles that change the stored data in the cell. These are known as soft errors [Chandra and Aitken 2009; Cai et al. 2006]. However, one of most dominant causes of SRAM cell failures is process variation [Mukhopadhyay et al. 2005; Djahromi et al. 2007]. It is well known that the process variations in the semiconductor are a significant concern for designers and this concern is only going to exacerbate for future technology nodes. Process variations affect various semiconductor process parameters such as channel length, oxide thickness, etc. These variations in process parameters can be inter-die or intra-die. Inter-die variations change the parameters in the same direction for all transistors within a single die. Intra-die variations, on the other hand, cause a mismatch between parameters of the transistors within a single die. These intra-die

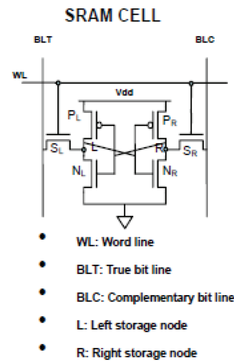


Fig. 4. 6-T SRAM cell (reproduced from Khajeh et al. [2009]).

variations may be systematic, that is, spatially correlated, or they may be random. Spatially correlated variations do not cause large mismatches between neighboring transistors [Mukhopadhyay et al. 2005]. Random intra-die variations, mostly caused by Random Dopant Fluctuations (RDF), lead to mismatches between neighboring transistors and are the dominant cause of failures in SRAM cells [Mukhopadhyay et al. 2005; Djahromi et al. 2007; Calhoun and Chandrakasan 2006]. Though process variation can affect many different process parameters, its influence can be effectively lumped into variation of threshold voltage Vth for individual transistors.

Figure 4 shows the typical six-transistor cell used in CMOS SRAM. During the read operation, the read access time is very sensitive to the variations in the threshold voltages of the access transistors (SR or SL) and the pull-down transistors (NR or NL) depending on the stored value. During the write operation, threshold voltage variations of the access transistors and the pull-up transistors (PR or PL) have the strongest effect on the write time. SRAM cell failures induced by process variations are also known as parametric failures. Parametric failures can be of different types and are described as follows [Makhzan et al. 2007].

(a) *Read Access Failure*. A reduction in bitline voltage differential during a read operation within the maximum allowed time is called a read access failure.
(b) *Write Access Failure*. This is an unsuccessful write within the maximum allowed time.
(c) *Read Stability Failure (Destructive Read Failure)*. An increase in the PMOS/NMOS node voltage beyond the trip voltage of the inverter pair causing a bit flip during a read operation is known as a read stability failure or destructive read failure.
(d) *Hold Failure*. A bit flip while in standby mode caused by decrease in data retention voltage is referred as a hold failure.

Our proposed MC^2 architecture is effective against all of the preceding SRAM failures.

Dynamic nature of SRAM failures. As shown by Khajeh et al. [2009], the probability of SRAM failure is highly dependent on Vdd, frequency of operation, and temperature. It is well known that a decrease in Vdd increases the probability of SRAM failures [Djahromi et al. 2007; Mukhopadhyay et al. 2005]. However, an increase in Vdd increases the dynamic and leakage power dissipation, which in turn increases the temperature. Increase in temperature causes an increase in cell delay, resulting in a higher probability of failure. Moreover, the increase in temperature increases the leakage power further, resulting in a positive feedback loop between the two [Khajeh et al. 2009]. Therefore, as Vdd is increased, temperature as well as leakage and dynamic

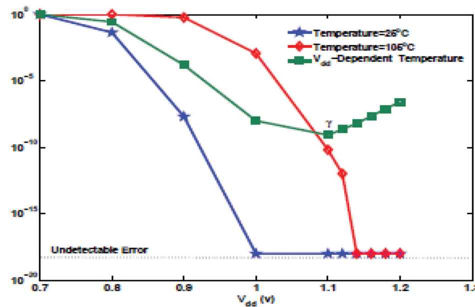


Fig. 5. Unexpected increase in failure as Vdd is increased due to interaction between leakage power and temperature (reproduced from Khajeh et al. [2009]).

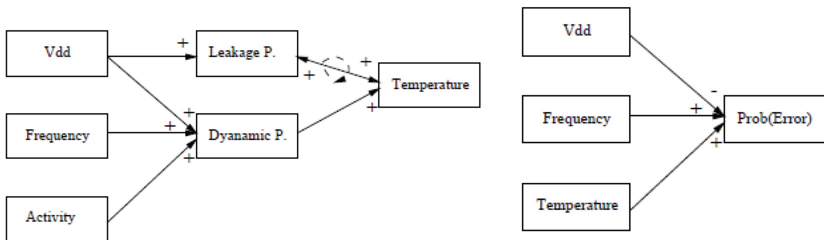


Fig. 6. Interrelationship of Vdd, frequency, temperature, leakage/dynamic power, etc., and their effect on SRAM failure probability (based on Khajeh et al. [2009]).

powers may increase in an interdependent fashion, increasing the failure probability unexpectedly, as shown in Figure 5. Additionally, if a set of SRAM cells is accessed very frequently or if they are located near hotspots such as execution units, the dynamic power dissipation of these cells will increase, leading to an increase in temperature and inducing the cells towards failure [Meterelliyoz et al. 2008]. Figure 6 (based on Khajeh et al. [2009]) pictorially depicts the relationships between Vdd, frequency, temperature, memory activity, power dissipation, and probability of failure. From the previous discussion, we note that SRAM failures are very dynamic: not only affected by user-controlled operating conditions such as Vdd and frequency, but also by other conditions such as voltage irregularities, leakage, temperature, nearby hotspots, memory access pattern, and drift in frequency—all of which may be constantly changing and are beyond the control of the user. It is therefore crucial that any error control mechanism in SRAM caches be responsive to dynamic changes in error patterns, without being dependent on static error maps. MC^2 accomplishes this.

3. MC^2 ARCHITECTURE

3.1. Basic Mechanism

The basic idea behind multicopy cache (MC^2) is to maintain multiple copies of each data item in the cache. Such a mechanism makes the cache resilient to a high number of SRAM failures. As long as the same bit-position of every copy is not affected by failures, the errors can always be detected and may also be corrected[1]. This high error resiliency technique allows the cache to operate under aggressively low Vdd, leading to reduction

[1]It is very rare that the same bit-positions of more than one copy will be affected by failures. For a bit failure rate of 10^{-6}, the failure rate for two copies of a 32-bit word is over 1 million times lower than that with a single copy.

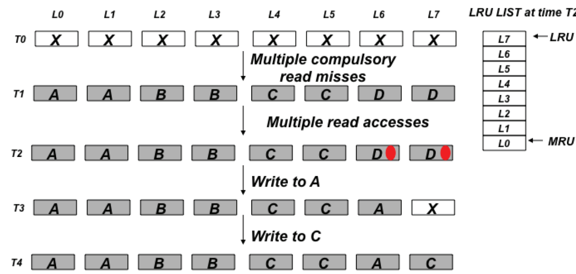


Fig. 7. Example 1 of reads and writes to MC^2.

in energy and power consumption. The multicopy mechanism may be implemented in many different ways, depending on cache organization, type of cache (instruction or data), write policy (write-through or write-back), write-miss policy (write-allocate or no write-allocate), and replacement policy of the cache; each implementation of MC^2 will incur some overhead and potentially some performance degradation. Thus the MC^2 architecture must be designed carefully to minimize these overheads while achieving low energy with high resiliency.

In this article, we present the Redundancy through Duplication and Triplication (RDT) policy for MC^2. We assume a write-back data cache with write-allocate policy and true LRU replacement policy. MC^2 with RDT policy maintains two copies of each clean data and three copies of each dirty data in the same set. If a data in the cache is clean (i.e., unmodified by the processor), a correct copy of that data is also available in the lower level of memory (LLM). Therefore only two copies of the clean data need to be maintained in the cache. Whenever the processor reads a clean data, both copies are compared with each other. If there is any mismatch, an error is detected and the requested data is read from the LLM and forwarded to the processor. On the other hand, if the requested data in the cache is dirty (i.e., modified by processor), the most updated version of that data is in the cache and not in the LLM. Hence for dirty data, three copies are maintained in the cache so that in the event of error(s), the correct data can be generated by majority voting logic using all three copies. Thus, the use of the RDT policy would result in two or three cache lines with same data in a given set.

Examples. Figure 7 shows the physical contents of a particular set s of an 8-way set-associative MC^2 for a sequence of reads and writes. Each of the eight rectangles represents a cache line in the given set. The top horizontal labels *L0* to *L7* indicate indices of the physical cache lines. The vertical left labels *T0* to *T7* indicate timestamps. Initially, at time T0, the cache is cold and the set empty. Then four cache lines *A*, *B*, *C*, and *D*, belonging to set s, are read one after another from the memory and placed in the set. At that point, the set s contains four clean data, each with two adjacent copies. Assume, at time *T2*, that there are multiple read accesses such that *D* becomes the LRU data (marked by dots). The LRU list at time *T2* is also shown in the figure. Now the processor writes to data *A*. After the write is completed, *A* would become dirty, hence it would need three copies instead of its present two copies, requiring a new copy of *A* to be created. This is done by evicting both copies of LRU data *D* and using one of the freed cache lines to store the third copy of *A*. Next, the processor writes to data *C*. A new copy of *C* is created using the empty cache line present in the set s. As the example shows, all copies of a given data may not be physically adjacent. Figure 8 illustrates a variant of Example 1 in which, at time *T2*, the processor issues a write to data E, resulting in a write-miss and a write-allocate. The replacement algorithm replaces two

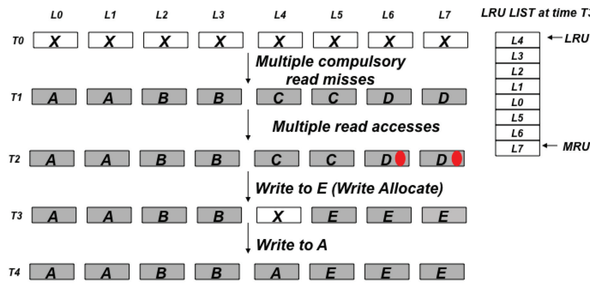


Fig. 8. Example 2 of reads and writes to MC^2.

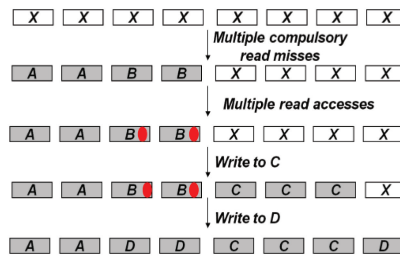


Fig. 9. Example 3 of reads and writes to MC^2.

lines D and C according to the LRU list and creates three copies of data E. The LRU list is updated accordingly.

Figure 9 shows another example. Again, assume the cache set s is completely empty in the beginning. The processor reads data A and B, which are placed in the set with two adjacent copies for each data. The processor reads these two data multiple times such that B becomes the least recently used data. Then the processor performs two writes to two separate cache lines, namely, data C, followed by data D. Since we assume a write-back cache with write-allocate policy, both writes would result in write-misses. Therefore, the write-miss for data C is served and three copies of cache line C are created in set s and updated with the data from the processor. At this point, there is only one empty cache line left in the set. Now, when the write-miss for data D is served, three copies of data D must be created. Since data B is the least recently used data, the cache lines belonging to data B along with the only empty cache line are used to store three copies of data D. We now describe the detailed MC^2 RDT cache architecture operation and implementation overheads.

3.2. Cache Architecture and Operation

In a conventional N-way associative cache (Figure 10), there are N tag comparators producing N way-select signals (WS1-N). For each cache access, at most one of the N way-select signals is true as there is a maximum of one copy of each line in the set. During cache read, the way-select signals drive the output multiplexer/driver that outputs the requested data from the selected way. During cache write, way-select signals are also used to write the data to the selected way. In MC^2 the tag comparison circuit remains unchanged. However, for the RDT policy, for any given access, at most three of the N way-select signals may be true. In the MC^2 architecture (Figure 11), the output multiplexer is replaced by data error detection and correction logic. Using the way-select signals, this logic compares multiple copies of the requested data and accordingly detects and corrects errors. It outputs the corrected data and the read error

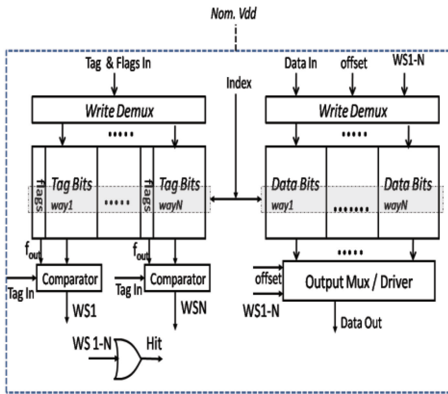


Fig. 10. Conventional cache architecture.

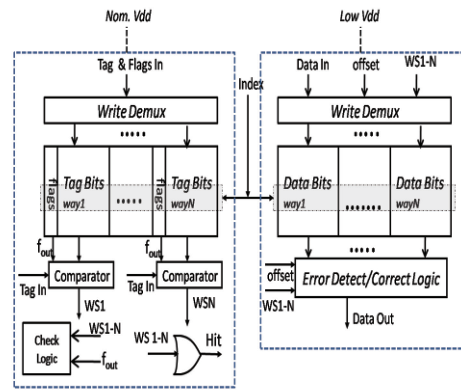


Fig. 11. MultiCopy Cache architecture.

signal, indicating whether there is any error in the accessed data. The write demultiplexer of the MC^2 allows simultaneous writing to multiple cache lines in the data and the tag array of the set. Additionally there may be an optional check logic that processes the cache flags (clean/dirty) and the way-select signals to ensure that the RDT policy is being followed (i.e., two copies for every clean data, three copies for every dirty data). The flags logic will raise a machine check interrupt if it finds any policy violation. In the CC, for each access, the LRU list is updated once to reflect read or write of maximum one physical cache line in the set. In MC^2, as shown in the examples, for each access, the LRU list is updated multiple times to reflect reads or writes of either two or three physical lines.

Furthermore, in CC, the entire cache is run on one Vdd domain, typically connected to the processor Vdd (nominal Vdd) for L1 caches. Similarly, in MC^2, the tag side of the cache (tag SRAM array, comparators, decoder, and write demultiplexer) is connected to a nominal Vdd. However, MC^2's data side (data array, error detect/correct logic, decoder, and write demultiplexer) is run on a separate Vdd that can be aggressively scaled down. Level shifters are required while going from low to high Vdd only (i.e., for data-side output to processor) [Diril et al. 2005] and generate negligible overheads as shown in Sections 5 and 6.

Figure 12 shows the flowchart of the MC^2 RDT cache operation. The address tag bits from the processor are compared with the tag bits stored for each cache line of the selected set. The RDT policy ensures that, for any given address, there can be: no tag match (cache miss); exactly two matches (cache hit for clean data); exactly three matches (cache hit for dirty data). In case of a miss (referred to as *conventional miss*), a new cache line is fetched from the LLM. Depending upon the type of access (read or write), either two or three cache lines in the selected set are evicted and replaced by the newly fetched line. For write accesses, all three copies must be updated. For cache hits for clean data, the type of access may be either read or write. For read accesses, both copies of the clean data are compared with each other. If there is a match, the data is forwarded to the processor. This is referred to as *clean read hit*. If there is no match, a *read error miss* has happened and the correct copy of the requested data is fetched from LLM. Write accesses to any clean data always result in *clean write misses*. This event represents clean to dirty transition (top right corner of Figure 10). The cache line corresponding to the accessed address is fetched from the LLM. An existing line in the selected set is evicted and replaced with the newly fetched line, resulting in three copies of the accessed data. Now all three copies are marked dirty and updated with the data from the processor. If there are three tag matches during tag comparison, this

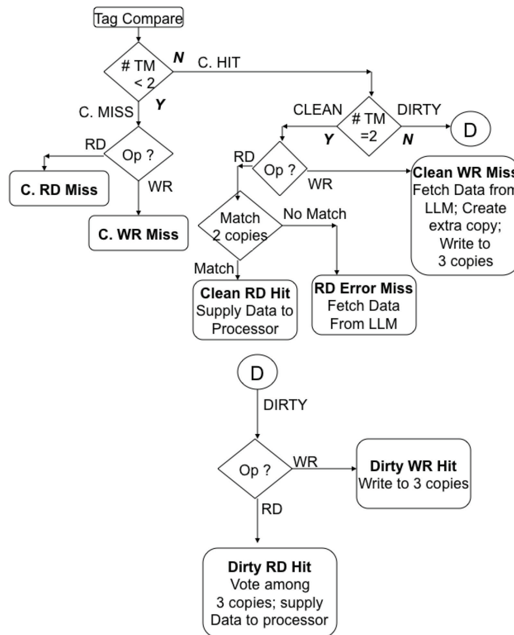


Fig. 12. Flowchart showing operation of MC^2 RDT cache (TM represents "Tag Matches").

must be a cache hit for dirty data. If the access is a read operation, majority-voting logic among all three copies generates the correct data. This is referred to as *dirty read hit*. If the access is a write operation, all three copies must be updated with the data from the processor. This is referred to as *dirty write hit*. Whenever there is a dirty data write-back, all three copies are identified using a secondary tag search and processed; then the correct data is written to LLM.

Dirty Data Write-back. Whenever there is a conventional miss or a clean write-miss, one, two, or three cache lines may be replaced with data from LLM. Using a true LRU policy, up to three cache lines may be evicted in a single cache access (for example, a conventional write-miss). If any of the replaced lines is dirty, it must be ensured that the correct data is being written to the memory. Therefore, whenever a dirty line is being evicted from the cache, all other copies of that dirty line must be located and processed before writing to the LLM. This is done by accessing the tag array and performing another set of comparisons with tag bits of the dirty cache line being replaced. After locating all three copies, they are read into a write-buffer. The write-buffer will store all three copies of the dirty data if the bus is not available immediately. The write-buffer will perform a majority-voting logic using all three copies and write the correct data to the LLM, whenever the bus is available. Irrespective of the size of the write-buffer, only one majority-voting logic is required, namely, only for the data in the top of the write-buffer.

The MC^2 RDT cache will incur overheads in delay, area, and energy due to error detect/correct logic, check logic, level shifters, and additional operations required during cache access (such as writing to multiple lines and tags). Sections 5 and 6 examine these overheads and establish that these are minimal.

4. RELATED WORK

There are several previous works related to improving SRAM reliability in the face of process variation and soft errors, especially for low-voltage operation. A number of

these works approach the problem from a circuit perspective, improving reliability of each SRAM cell. Apart from the familiar 6-T SRAM cell, 8-T SRAM cell [Chang et al. 2005] and 10-T SRAM cell [Calhoun and Chandrakasan 2006] have been proposed. Both 8-T and 10-T SRAM cells improve read stability, though the stability of the inverter pair remains unchanged. Kulkarni et al. [2007] proposed a Schmidt-trigger-based 10-T SRAM cell with inherent tolerance towards process variation using a feedback-based mechanism. However, this SRAM cell requires a 100% increase in area and about 42% increase in access time for low-voltage operation.

Several architectural techniques have also been proposed to improve reliability of on-chip cache by using redundancy. It is typical in the industry to have redundant rows and columns in the SRAM cache. Any defective row or column may be detected before shipping and is replaced by a redundant row or column using laser fuses [Schuster 1978]. However, although this technique is effective against manufacturing defects, it is not against process-variation-induced errors that depend heavily on operating conditions such as Vdd. A number of other techniques have been proposed to improve SRAM array reliability against process variation failures. Wilkerson et al. [2008] proposed multiple techniques using part of a cache line as a redundancy for defective bits for the rest of the cache lines in the same set. It disables the faulty words and replaces them with nonfaulty ones in the same set. Agarwal et al. [2005] proposed a fault-tolerant cache architecture in which the column multiplexers are programmed to select a nonfaulty block in the same row if the accessed block is faulty. A similar work is PADed caches [Shirvani and McCluskey 1999], which uses programmable address decoders that are programmed to select nonfaulty blocks as replacements of faulty blocks. Makzhan et al. [2007] and Sasan et al. [2009a, 2009b] proposed a number of cache architectures in which the error-prone part of the cache is fixed using either a separate redundancy cache or parts of the same cache, or using charge pumps to increase Vdd of the defective wordlines. However, all the techniques proposed in these works [Wilkerson et al. 2008; Agarwal et al. 2005; Makhzan et al. 2007; Sasan et al. 2009a, 2009b; Shirvani and McCluskey et al. 1999] require BIST characterization of the cache and generation of some form of a cache error map with various levels of granularity: per wordline, per cache line, per byte, etc. Whenever Vdd is scaled up or down, the BIST engine is run and the entire cache memory is characterized, generating an error map. Every time BIST characterization is run, the cache has to be flushed of its current contents followed by writing, reading, and comparing by the BIST engine before the cache is ready for use. The time overhead of the BIST characterization would limit the frequency at which Vdd can be scaled up or down. The storage of the error map, depending upon its granularity, also increases the area overhead of the cache. Even with these costs, a basic assumption behind the preceding works is that, once the BIST characterization is done, the error map perfectly describes the locations of process variation errors until the next change in Vdd. As discussed in Section 2, this assumption is not valid because of the dynamic nature of SRAM failures.

In order to improve SRAM reliability against such dynamic failures, dynamic error detection and correction ability is required without using any static error map. One of most popular mechanisms for such dynamic error detection/correction is Error Control Coding (ECC), widely used in caches and memories. The simplest form of ECC is one-bit parity, which detects odd numbers of errors in the data and is often used in L1 caches [Genua 2004]. Since such one-bit parity mechanisms do not have any correction capability, they are not useful except for instruction caches or data caches with a write-through policy. Another form of ECC used in caches is Single Error Correction, Double Errors Detection (SECDED). Hsiao et al. [1970] proposed an optimal minimum-odd weight column SECDED code suitable for fast implementation in memory. However, despite its optimality, the Hsaio code incurs multiple clock-cycle latencies for caches and significant area overhead (about 30%), as we show later in the results. Kim et al.

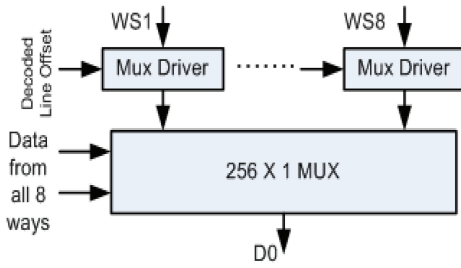


Fig. 13. Output multiplexer/driver of conventional cache (16K 8-way cache).

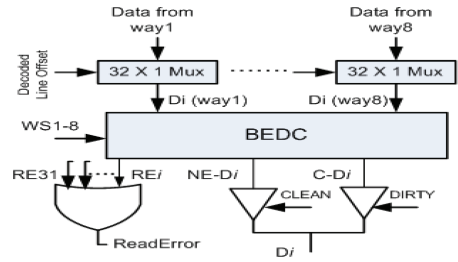


Fig. 14. Data error detection and correction logic of MC^2 (16K 8-way cache).

[2007] and Naseer and Draper [2008] have proposed ECC schemes to correct multiple error bits and further improve reliability beyond SECDED. However, as shown by Mazumder [1993] and Agarwal et al. [2005], an ECC mechanism beyond one-bit correction capability cannot be implemented in memory because of the area and delay overheads. Zhang et al. [2003] used replication of some "hot" frequently used cache lines to mitigate soft errors. Such a technique has nonuniform error tolerance and is ineffective towards process-variation-induced failures. Moreover, it requires an additional error control mechanism such as parity for its operation. A recent work uses the configurable part of the cache for storing multiple ECC check bits for different segments of the cache line using an elaborate orthogonal Latin Square code ECC [Chishti et al. 2009] to enable dynamic error correction. This requires up to eight levels of XOR gates for decoding, resulting in a significant increase in cache critical-path delay.

In contrast to previous works, our MC^2 architecture is able to detect and correct errors dynamically without requiring any BIST characterization and error map storage. MC^2 provides better reliability than the SECDED cache with minimal area overhead and much less latency to detect/correct errors. Compared to elaborate ECC mechanisms such as Chishti et al. [2009], the MC^2 architecture has a simple detection/correction mechanism resulting in much lower delay overhead. Indeed, as we show in the next section, MC^2 incurs less than 3% overhead in cache delay and less than 2% increase in read hit dynamic energy with negligible area overhead.

5. HARDWARE IMPLEMENTATION AND OVERHEADS

The MC^2 architecture has three main changes over a Conventional Cache (CC): (a) the output mux of the CC (Figure 13) is replaced by the data detection and correction logic (Figure 14); (b) level shifters are needed only when signals travel from low- to high-Vdd domains (data-side output to the processor); and (c) additional operations are required during cache access (writing to multiple lines and tags during cache write, extra line-fill during clean write miss). For level shifting, we use dual Vdd/Vth logic gates with built-in level shifting [Diril et al. 2005] at the data-side output of the cache. Such gates use a higher threshold voltage for PMOS transistors driven by low-Vdd inputs and have only a slightly increased delay ($<$10ps) and almost no additional overhead in power consumption compared to conventional single-Vdd gates.

The data error detection and correction logic is formed by a number of *Bit Error Detect/Correct (BEDC)* logic blocks, one for each data bit (Figure 14). BEDC$_i$ for the ith data bit outputs the following signals: (a) RE$_i$, read error signal, (b) NE-D$_i$, no-error data bit, (c) C-D$_i$, corrected data bit. RE$_i$ indicates whether there is any error and is relevant only for clean data. NE-D$_i$ is the correct output bit if there is no error while C-D$_i$ is the corrected output bit selected using majority voting of three copies. The BEDC block essentially consists of two parallel parts, namely one for error detection

and the other for error correction. The error detection logic uses N-input OR and AND gates to produce outputs RE_i and NE-D_i. The error correction logic uses an N-input XOR gate to produce output C-D_i (Figure 15). To enable quantitative comparisons, we synthesized the output logic for MC^2 (error detect/correct logic and the check logic) as well as the multiplexer and output driver of CC using Synopsys Design Compiler for a TSMC 65nm typical library for a 16K 8-way associative cache. We found that the delay of MC^2 output logic is increased only by 5% compared to the CC output mux while area and power consumption are lower than the CC mux. The output driver of CC, implemented as specified in Tarjan et al. [2006], uses one multiplexer driver for each way; each multiplexer driver drives 32 tri-state buffers, resulting in 256 tri-state buffers for each output bit. However, in MC^2 there are only eight 32X1 multiplexers for all output bits along one BEDC block for each output bit. This results in the CC output mux incurring higher area and power compared to MC^2 output logic.

To put these overheads in perspective of the entire cache, we used CACTI 4.1 [Taijan et al. 2006] and estimated that, for a 16K 8-way associative cache, the output mux/driver contributes to 48% of total delay, 8% of dynamic power, 10% of leakage power, and 3% of area for CC. Recall that the MC^2 architecture replaces CC's output mux with MC^2 output logic; we expect that the multicopy mechanism will increase the delay of the cache nominally (estimated <3%). Though the area of MC^2 output logic is found to be less than the CC mux area, we recognize CC mux implementation may be further optimized and hence we conservatively conclude that MC^2 has no appreciable area overhead.

$$RH = t_R + d_R \tag{5.1}$$

$$WH = t_R + d_{W,1w} \tag{5.2}$$

$$RLF = t_{W,1} + d_{W,1l} + m_R \tag{5.3}$$

$$WLF = t_{W,1} + d_{W,1l} + m_R \tag{5.4}$$

$$WB = d_{R,1l} + m_W \tag{5.5}$$

$$RM = t_R + RLF + WB \tag{5.6}$$

$$WM = t_R + WLF + WB \tag{5.7}$$

$$RLF = t_{W,2} + d_{W,2l} + m_R \tag{5.8}$$

$$WLF = t_{W,3} + d_{W,3l} + m_R \tag{5.9}$$

$$WB = d_{R,3l} + m_{W,C} \tag{5.10}$$

$$ELF = t_{W,1} + m_R + d_{W,1l} + d_{W,3w} \tag{5.11}$$

$$RM = t_R + fl + RLF + WB \tag{5.12}$$

$$WM = t_R + fl + WLF + WB \tag{5.13}$$

$$CRH = t_R + fl + d_{R,1w} + l_{CORR} \tag{5.14}$$

$$REM = t_R + fl + d_{R,1w} + l_{CORR} + m_R \tag{5.15}$$

$$CWM = t_R + fl + ELF + WB \tag{5.16}$$

$$DRH = t_R + fl + d_{R,1w} + l_{CORR} \tag{5.17}$$

$$DWH = t_R + fl + d_{W,3w} \tag{5.18}$$

In order to account for additional operations in MC^2, we developed an analytical dynamic energy model for each cache event. In a conventional cache, a cache access

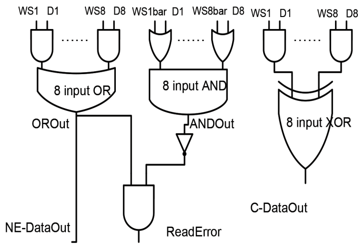


Fig. 15. Bit Error Detection and Correction (BEDC) logic.

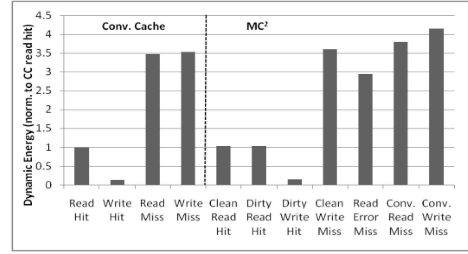


Fig. 16. Normalized dynamic energy of CC and MC^2 events (all normalized to CC read hit).

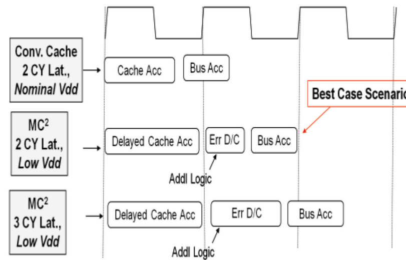
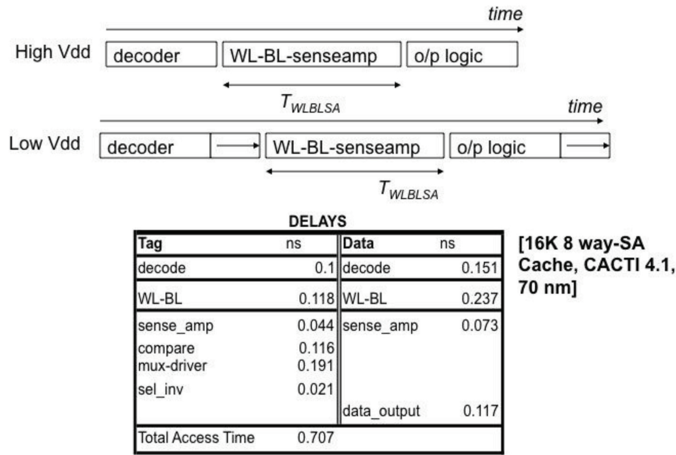


Fig. 17. CC and MC^2 latencies.

results in one of the following four mutually exclusive events: (1) *Read Hit* (RH); (2) *Write Hit* (WH); (3) *Read Miss* (RM); and (4) *Write-Miss* (WM). Each of these events consists of several micro-operations such as tag array read, data array read, Read miss Line-Fill (RLF), Write-miss Line-Fill (WLF), dirty data write-back (WB). Eqs. (5.1)–(5.7) describe the energy model used for each of these events for the conventional cache. For the MC^2 architecture, a cache access results in one of the following seven mutually exclusive events: (1) *conventional Read Miss* (RM); (2) *conventional Write-Miss* (WM); (3) *Clean Read Hit* (CRH); (4) *Clean Write-Miss* (CWM); (5) *Read Error Miss* (REM); (6) *Dirty Read Hit* (DRH); and (7) *Dirty Write Hit* (DWH). As explained in Section 3, for CWM, an extra copy of the existing data is brought into the cache, followed by a write to all three lines. This is represented as *Extra-Copy Line-Fill* (ELF). Eqs. (5.8)–(5.18) describe the energy model of the events for the MC^2 architecture.

Individual cache micro-operations referred to in the preceding equations are described here.

t_R: *Tag Read*. Read all tags for the selected set and compare with the input tag.
d_R: *Data Read*. Read data for all ways in parallel for the selected set.
$d_{R,1l}$: *Data Read (1 line)*. Read a given cache line in the set.
$d_{W,iw}$: *Data Write (i words)*. Write i words of data to the specified way of the selected set.
$t_{W,i}$: *Tag Write(i)*. Write i tags of the selected set.
$d_{W,il}$: *Data Write(i lines)*. Write i cache lines of data to the specified way of the selected set.
m_R: *Mem Read*. Read lower level of memory (including bus access).
m_W: *Mem Write*. Write to lower level of memory (including bus access).
fl: *Flags Logic*. Check if number of copies matches status of the line (dirty or clean).
l_{CORR}: *Data Corr Logic*. Perform data error detection and correction.
$m_{W,C}$: *Mem Write (with correction)*. Write to LLM after applying majority-voting logic.



Tag	ns	Data	ns
decode	0.1	decode	0.151
WL-BL	0.118	WL-BL	0.237
sense_amp	0.044	sense_amp	0.073
compare	0.116		
mux-driver	0.191		
sel_inv	0.021		
		data_output	0.117
Total Access Time	0.707		

Fig. 18. Cache access pipeline.

We used CACTI 4.1 [Tarjan et al. 2006] to obtain energy consumptions of the individual micro-operations involved in the events discussed before for a conventional cache and an MC^2 cache. The results of this model are shown in Figure 16. For example, there is only 2% increase in dynamic energy for read hit and about 17% increase for write-miss. Since there is only a negligible area overhead, we can safely assume there is no change in the leakage power. It should be noted that the total energy overhead of MC^2 is application dependent. We examine this aspect in more detail in the next section.

As shown in the figure, the access delay of MC^2 is increased to less than 3% of the access delay of the CC due to error detection and correction logic. In addition, as Vdd is lowered, only the logic stages of the cache, that is, the decoder stage and the output logic stage, would experience an increase in delay according to the equation $delay \propto Vdd/(Vdd - Vth)^{1.3}$. It is to be noted that the time delay for accessing wordlines, bitlines, and sense amplifiers will remain unchanged as Vdd is lowered. The probability of bit failure will increase, as discussed in Section 6.1. Figure 18 shows the pipeline stages of the cache and the estimated delay of each stage, as obtained from CACTI 4.1 for a 16K 8-way associative cache for 70nm. Recall that in MC^2, we scale down the Vdd of only the data side of the cache. The logic stages of the data side of the cache amounts to 268ps. Considering Vth = 0.25V, if the Vdd is scaled down from nominal 1.1V to 0.55V, the logic delay will increase by 93%. If we assume a reasonable frequency of 1 GHz, the increase in logic delay due to Vdd scaling amounts to less than one-quarter of a cycle time. The increased logic delays may be further reduced using dual-Vth caches [Mamidipaka and Dutt 2004]. In general, the impact on the cache access timing would vary depending on a number of factors such as cache associativity, organization, logic implementation, and process (technology node and use of dual Vth). Hence, rather than examining random design configurations, we evaluate the MC^2 architecture for the design corner cases, that is, the best- and the worst-case timing scenarios. The load/store latency of two cycles is assumed to be broken into actual cache access takes place in cycle 1, while the bus access takes only a part of the next cycle. Based on this, we assume two scenarios for MC^2 cache access, as shown in Figure 17: (a) a best-case (MC^2-B) MC^2 delay overhead including the increased logic delay due to Vdd scaling fits in the remaining time in the second cycle, resulting in total 2-cycle latency and no penalty; (b) a worst-case (MC^2-W) error detection/correction delay along with increased logic delay at the lowest Vdd of operation is long enough such that a total of three cycles

Table II. Base Processor Configuration (ARM-11-like)

I-cache	16KB, 2 cycle
L2 cache	256KB, 15 cycles
Fetch, dispatch	1 wide
Issue	In-order, non blocking
Execution	Out-of-order
Memory	30 cycles
Instr Fetch Queue	4
Ld/Str Queue	16
RUU size	8
Execution units	1 INT, 1 FP simple & mult/div
Pipeline	8 stages
Frequency	1 GHz

Table III. Benchmarks (all with large input sets from MiBench)

Automotive	basicmath, bitcnt, qsort, susan (smooth, edges, corners)
Consumer	jpeg-encode, jpeg-decode, lame, mad, tiff2bw, tiff2rgba, tiffdither, tiffmedian
Networking	dijkstra, patricia
Office	ghostscript, stringsearch
Security	sha, pgp.sign, pgp.verify
Telecom	crc32, fft, ifft, gsm-encode

is needed for every access. It is to be noted that as Vdd is scaled down in the data side of MC^2, clock frequency will remain unchanged, since the effect of the increased logic delay is already considered in the best-case and worst-case access latencies discussed before. The dynamic energy overhead of MC^2 depends on cache miss energy, assumed to be five times the cache hit energy according to Huang et al. [2001]. As discussed in Section 3, the LRU list must be updated multiple times for each access to MC^2. A true LRU implementation uses an associative memory of $(m-1)\ {}^*log_2 m$ bits [Roy 2009] for an m-way associative cache, that is, 21-bit memory for an 8-way associative cache. We expect the delay of additional updates to the LRU structure to be completely hidden by cache data memory access and to contribute negligible overhead to dynamic energy.

6. EXPERIMENTAL RESULTS

The experiments are designed to evaluate the MC^2 architecture in terms of energy and performance for standard embedded benchmarks.

Tables II and III outline our experimental setup for the base processor configuration and benchmarks, respectively. The processor is ARM-11-like configured with a 16K 8-way set-associative cache. We modified Simplescalar 3.0 [Austin et al. 2002] extensively to support the MC^2 architecture. The embedded benchmarks are from the MiBench suite [Guthaus et al. 2001]. All benchmarks are compiled with Compaq alpha compiler using an –O4 flag for Alpha 21264 ISA.

We carried out the following detailed experiments to evaluate the MC^2 architecture.

(1) *Circuit Simulation and Analytical Calculations.* We carry out SPICE simulations to measure probability of failure for an SRAM cell, followed by analytical calculations to determine the probability of failure for conventional cache and MC^2 architectures.
(2) *Comparison with Conventional Cache at Nominal Vdd.* We compare performance and energy consumption of a conventional cache (CC) running at nominal Vdd with that of an MC^2 running with scaled Vdd for two different SRAM cells.
(3) *Comparison with Conventional Cache at Different Yields.* We compare a conventional cache with the MC^2 architecture, each operated at its minimum Vdd for a given set of yields.
(4) *Miss-Energy-Sensitivity Analysis.* We carry out a sensitivity analysis by increasing the energy of cache misses and observe the impact on MC^2 in comparison to a conventional cache.
(5) *Comparison with Other ECC Caches.* We compare a conventional cache with and without a traditional ECC mechanism (SECDED) versus the MC^2 architecture. All

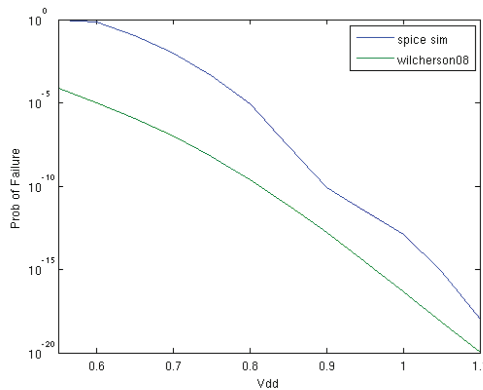


Fig. 19. Probability of failure vs. Vdd for SRAM cell from SPICE simulation and from Wilkerson et al. [2008].

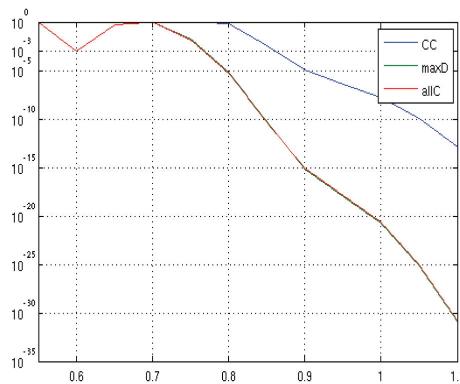


Fig. 20. Probability of failure vs. Vdd for a 16KB cache for SRAM cell B.

three caches are constrained to have almost the same area and the same failure rate.

(6) *Comparison with Voltage Frequency Scaling.* Finally, we compare voltage frequency scaling on a conventional cache with voltage scaling with fixed frequency on an MC^2 architecture.

6.1. Circuit Simulation and Analytical Modeling

We carried out a Monte Carlo SPICE simulation using Predictive Technology Models (PTM) [Zhao and Cao 2007] for 65nm with read/write access time of 250ps. The results show a drastic increase in failure probability due to process variation as Vdd is scaled down. The general trend of this data is corroborated with the SRAM failure probability data for 65nm used by Wilkerson et al. [2008], as shown in Figure 19. Based on these two sets of SRAM cell failure data from Wilkerson et al. [2008], we used analytical models of failure, as shown in Eqs. (6.1)–(6.4), to estimate the probabilities of failure for CC and MC^2 with 16KB size. The CC fails if there is one or more bit failures in the entire cache. For MC^2, the probability of failure of each data item depends on whether it is clean (i.e., having two copies) or modified (i.e., having three copies). A clean data in MC^2 fails if there is a bit failure in the same bit-position of both copies. A dirty data in MC^2 fails if there is a failure in two out of three bits occurring in the same bit-position of the three copies. Probability of failure of the entire cache depends on the numbers of

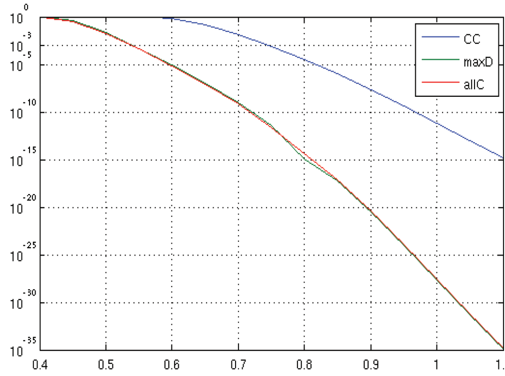


Fig. 21. Probability of failure vs. Vdd for a 16KB cache using SRAM cell A.

clean and dirty data in the cache. In Eqs. (6.1)–(6.4), $p = p(V)$ = probability of failure of each SRAM cell at a given voltage V, and N = number of SRAM cells, n = data bitwidth, *ncw* and *ndw* are the numbers of clean and dirty words, respectively, in MC^2.

$$\textit{Probability of failure of CC} = pfcc\,(V) = 1 - (1 - p)^N \tag{6.1}$$

$$\textit{Probability of failure of each clean word} = pfcw\,(V) = \sum_{i=1}^{n} \binom{n}{i} p^{2i}(1-p)^{2n-2i} \tag{6.2}$$

$$\textit{Probability of failure of each dirty word} = pfdw\,(V) = 1 - (1 - 2p^2 + p^3)^n \tag{6.3}$$

$$\textit{Probability of failure of } MC^2 = pm(V) = 1 - (1 - pfcw)^{ncw}\,(1 - pfdw)^{ndw} \tag{6.4}$$

The failure probability of CC and MC^2 is shown under two conditions: (a) *all clean*, where all data in the cache are clean; and (b) *maximally dirty*, where the cache has the maximum possible number of dirty data. The results clearly show that the MC^2 architecture achieves significantly higher reliability than CC. For the SRAM cell used by Wilkerson et al. [2008], the minimum Vdd (Vddmin) at probability of failure of 1e-3 is 0.55V, while for the SRAM cell used for our SPICE simulation the Vddmin for the same failure probability is 0.75V. Henceforth, the former SRAM cell is referred to as SRAM cell A, while latter is referred to as SRAM cell B.

Relationship with manufacturing yield. The manufacturing yield of a chip may be defined as the probability of failure of the entire chip at the recommended conditions (e.g., nominal Vdd, temperature, etc.). It depends upon the probabilities of different types of failures (such as parametric failures, soft errors, and hard failures) of the cache as well as the noncache part of the chip. Since, in this work, we are interested specifically in parametric failure of the cache and its impact on energy and performance, we use the probability of failure of the cache as a measure of yield.

6.2. Comparison with Conventional Cache at Nominal Vdd

Recall that a Conventional Cache (CC) is tied to the processor's nominal Vdd (Figure 10) whereas for MC^2, the data side can exploit voltage scaling (low Vdd in Figure 11). In this experiment, for SRAM cell A, we scale down the data-side Vdd of MC^2 until reaching 0.55V (for a failure rate of 10^{-3}) and measure IPC loss and reduction in total energy (dynamic and leakage), measured with respect to performance and energy of a CC at nominal Vdd. This is assumed to be 1.1V for a 65nm LSTP process according to ITRS [2008].

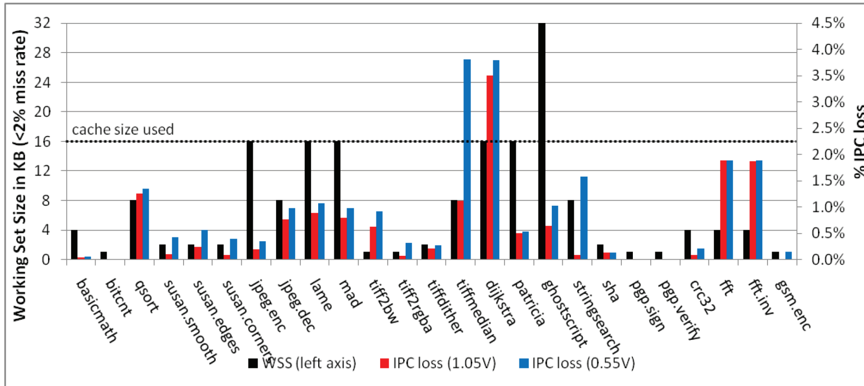


Fig. 22. Working-set size for each benchmark and % IPC loss (for 1.05V, 0.55V) with 16KB MC^2.

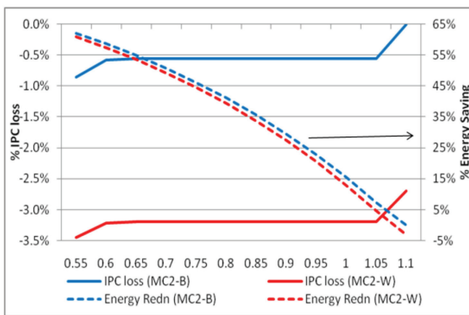


Fig. 23. IPC loss and energy savings of MC^2 vs. Vdd (with respect to CC at 1.1V) (average of all benchmarks) (SRAM A).

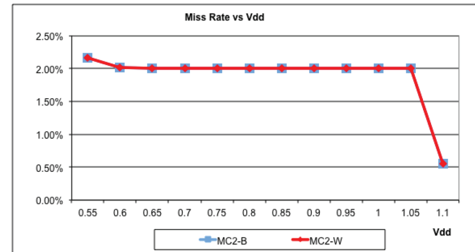


Fig. 24. Average miss rate vs. Vdd (SRAM A).

Figure 22 shows the Working-Set Size (WSS) for each benchmark (as explained in Section 2) and corresponding IPC loss for MC^2-*B* (best case) at 1.05V and 0.55V. As expected, benchmarks with low WSS (<=8KB) show low IPC degradation (<2% at 1.05V). This is due to a decrease in effective cache size/associativity in MC^2. Some benchmarks with even high WSS (>=16KB) (e.g., *jpeg-encode, lame, ghostscript*) have relatively low performance loss because of latency hiding due to out-of-order execution. When Vdd is further scaled down to 0.55V (inducing a large number of SRAM errors), some benchmarks (e.g., *tiffmedian* and *stringsearch*) experience high reduction in IPC relative to 1.05V, while some others (e.g., *dijkstra, fft*) have almost the same IPC at 1.05V. This is because each benchmark is affected differently depending upon the actual location of the errors and the memory access pattern.

Figure 23 shows performance degradation and energy savings, averaged for all benchmarks, as Vdd is scaled down: note that we observe a modest reduction in IPC, but rapid reduction in energy consumption. At lowest Vdd, the average IPC losses for MC^2 are 1% and 3.5% for the best and worst cases, respectively, while the corresponding reductions in energy are 61.5% and 59.5%. The average miss rate increases from 0.5% at 1.1V to about 2.1% at 0.55V, as shown in Figure 24. We repeated the preceding experiment using SRAM failure data obtained using SRAM cell B. The results, as shown in Figure 25, follow the trend observed previously. The average loss in IPC is about 2–4.5% at Vddmin and the reduction in EDP is about 45%. Thus, the MC^2 architecture results in high reduction in energy with low performance degradation.

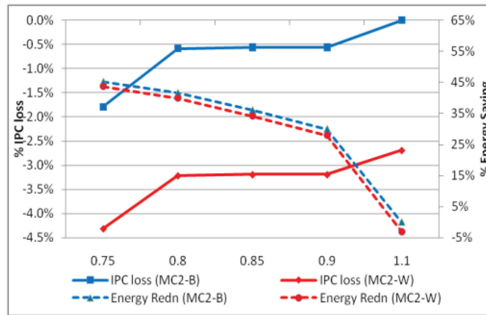


Fig. 25. IPC loss and energy savings of MC^2 vs. Vdd (with respect to CC at 1.1V) (average of all benchmarks) (SRAM B).

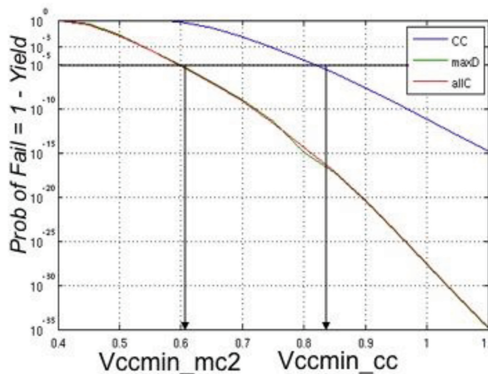


Fig. 26. Vccmin for CC and MC^2 using analytical models.

6.3. Comparison with Conventional Cache at Different Yields

In this experiment, we operate both the CC and MC^2 at their respective Vccmin for the target yield points. In this case, we assume that the L1 cache access is on the critical path of the processor. The nominal Vdd of the cache is constrained by its own access delay. Under such a condition, the cache Vdd can be scaled down to a level sufficient to ensure memory reliability according to the target yield point. Based on the probability of cache failures obtained using analytical models, the minimum Vdd for CC (Vccmin_cc) and minimum Vdd for MC^2 (Vccmin_mc2) are obtained for different yield points, as shown in Figure 26. The CC is operated at Vccmin_cc while the MC^2 is operated with its data side at Vccmin_mc2 and the tag side at nominal Vdd. We used the SRAM cell A [Wilkerson et al. 2008] for this experiment.

First, we observe that the difference in Vccmin for CC and MC^2 is significant at about 200–300mV (Figure 27). The difference increases as the target yield is increased. The performance loss for MC^2 is low at different target yield points; loss in IPC is less than 1% for MC2-B and less than 3.5% for MC2-W. On the other hand, energy and EDP for MC^2 are about 20–40% lower than CC depending upon the yield point (Figure 28). The results show that MC^2 achieves high reduction in energy with low performance loss for a wide range of yields.

6.4. Miss-Energy-Sensitivity Analysis

We expect that MC^2 will have more misses than a CC of the same size, because of the decrease in effective cache size and SRAM errors at low Vdd. Thus the energy savings

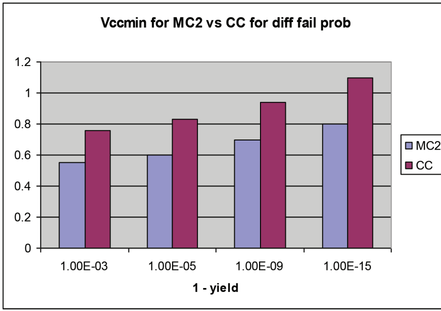


Fig. 27. Vccmin for CC and MC^2 for different failure rates.

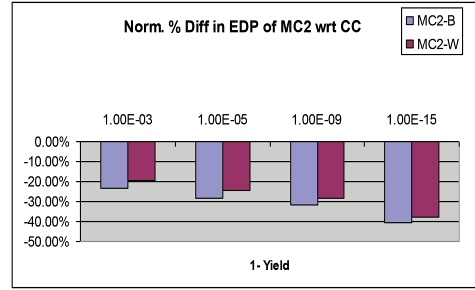


Fig. 28. Difference in EDP for MC^2-B and MC^2-W with respect to CC.

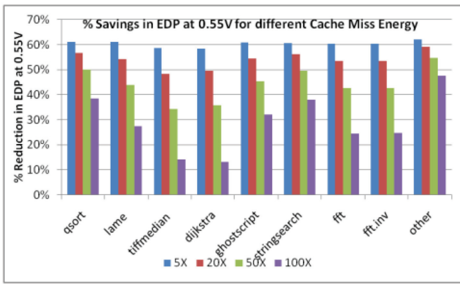


Fig. 29. Percentage savings in EDP at 0.55V for varying normalized miss energy.

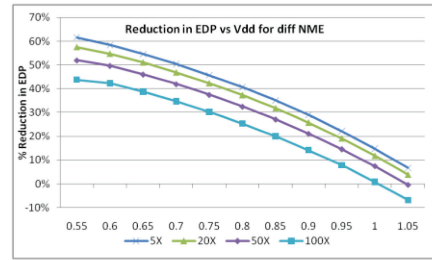


Fig. 30. Percentage reduction in EDP for different Vdd and different NME (average of all benchmarks).

from aggressive voltage scaling may be offset by increased energy from additional memory accesses due to higher miss rates. To study this phenomenon, we conduct a sensitivity analysis by varying the cache miss energy. We define normalized miss energy (NME) as the ratio of read miss energy to read hit energy for a CC of the same size. Energies of other miss events such as read error miss and clean write-miss are also increased proportionately. The NME depends on dynamic energy of accessing the LLM on cache misses. The dynamic energy of LLM, in turn, would be a function of the LLM architecture such as L2 cache size. In this experiment, NME is varied from 5–100. Though the exact value of NME depends on the implementation, typically for an L1 cache, an NME of 5 indicates an on-chip L2 cache [Huang et al. 2001], while an NME of 100 or more is likely if the L2 cache is off-chip [Zhang et al. 2005].

Figure 29 shows the percentage in reduction in Energy-Delay-Product (EDP) of MC^2 at 0.55V (measured with respect to CC at nominal Vdd) for the top eight benchmarks with the highest number of misses. We observe that for some benchmarks (e.g., *tiffmedian and dijsktra*), EDP increases significantly when the cache miss energy increases. We also measure the trend in EDP reduction, averaged over all benchmarks, for different Vdd and different NME, as shown in Figure 30. We observe that, even for high NME (50–100), MC^2 consumes less energy than CC when operated at Vdd 0.95V or lower. Therefore, for a wide range of NME and hence with both an off-chip and on-chip L2 cache, there exists a wide range of Vdd for which the energy and EDP of MC^2 are significantly less than those of CC at nominal Vdd.

6.5. Comparison with SECDED Cache with Equal Area

In this experiment, we compare MC^2 with SECDED ECC [Lin and Costello 1983], which is commonly used in caches for dynamic error detection and correction. Use of

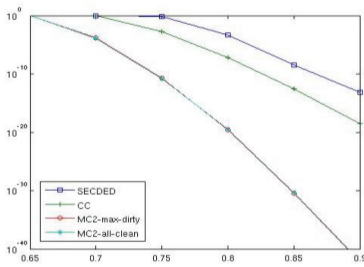


Fig. 31. Probability of failure vs. Vdd for SECDED cache, CC, and MC^2 under iso-area constraint.

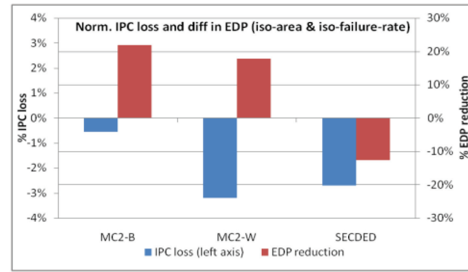


Fig. 32. Percentage IPC loss and percentage reduction in EDP (average of all benchmarks for MC^2 and SECDED cache, normalized to CC of equal area and equal probability of failure).

SECDED ECC for caches leads to significant area overhead because of extra parity bit storage (8 parity bits for 64 data bits) and associated logic. It also increases cache access timing significantly due to SECDED decoding logic on every read access. For a 16KB 8-way associative cache, based on synthesis results for 65nm and estimates from CACTI 4.1, SEDED ECC logic increases the delay of the output mux/driver by 135% and total cache delay by 65%. Area overhead of the SECDED cache is 45% over conventional cache without any ECC.

In order to carry out a fair comparison, we scale up SRAM transistor width such that the total area of the SECDED cache with smaller transistors is equal to the area of a CC and MC^2 with bigger transistors. We determined failure probabilities of both SRAM cells (bigger and smaller) using a Monte Carlo SPICE simulation with PTM models [PTM 2010] for 65nm. Then, we used our analytical models of failure to determine the failure probability of: (1) SECDED cache, (2) CC, and (3) MC^2, all with equal area. The results as shown in Figure 24, demonstrate that, at a given voltage, MC^2 is the most reliable cache whereas SECDED is the least reliable cache. In fact, we observe that the SECDED cache is worse than a CC of equal area for process-variation-induced failures. To explain this observation intuitively, SECDED needs additional bits and hence additional area to do error detection/correction. On the other hand, the same area may be used just to increase the size of the SRAM cell without using SECDE. Increasing the area of the SRAM cell slightly increases its reliability significantly, hence a cache with larger SRAM cells but no SECDED has more reliability than one with small SRAM cells with SECDED.

In order to compare performance and energy, we further constrain that all three caches (SECDED, CC, MC^2) have equal failure rate. Since SECDED is the least reliable, it is run at nominal Vdd 1.1V while aggressive voltage reduction is applied to CC and MC^2, such that the probabilities of failure of all three caches are same at the respective Vdd. We assume the data-side Vdd of CC can be scaled, like MC^2, as explained in Section 3. The latency of CC is assumed to be two cycles while that of the SECDED cache is assumed three because of the timing overhead of ECC logic. For MC^2, like previous experiments, we consider two different cases, namely, two cycles (best case: MC^2-B) and three cycles (worst case: MC^2-W), as discussed earlier.

Figure 25 shows the percentage of IPC loss and percentage difference in Energy-Delay-Product (EDP), averaged for all benchmarks, for MC^2 and SECDED cache when normalized to CC with equal area and at same probability of failure. We find the IPC loss for MC^2 is ∼0.5% for the best case (MC^2-B) and ∼3% for the worst case (MC^2-W). Performance of the SECDED cache is slightly less than that of the MC^2 worst case. The EDP of MC^2 is 30–35% lower than that of the SECDED cache. And MC^2 achieves

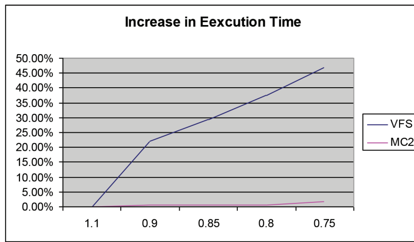


Fig. 33. Increase in execution time for VFS and MC^2.

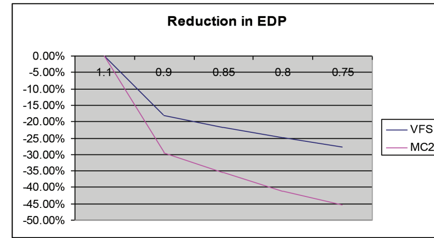


Fig. 34. Reduction in EDP for MC^2 and VFS.

greater than 20% reduction in EDP over CC, even when both are subject to aggressive voltage scaling for iso-failure rate. Hence, given an area budget and a failure rate, the SECDED cache will consume more energy than CC. We also establish that MC^2 would consume significantly less energy than CC with a low loss in performance.

6.6. Comparison with Traditional Voltage Frequency Scaling

In this experiment, we compare reduction in cache energy using traditional voltage frequency scaling with the energy reduction achieved in the MC^2 architecture as well as associated performance losses. In a conventional architecture, the cache shares the Vdd domain with the processor. In order to reduce cache power consumption, the Vdd is scaled down along with the increase in frequency to account for an increase in logic delay of the processor. However, in MC^2, the data side of the cache uses a Vdd domain separate from the rest of the system. Therefore, as discussed in Section 5, in MC^2, Vdd can be scaled down without increasing the frequency. We have shown that a 2-cycle access latency of a conventional cache may increase to maximum three cycles in the worst case for MC^2 for a very low operating Vdd (0.55V). We assume as in AbouGhazaleh et al. [2007] for traditional voltage frequency scaling, the frequency scales linearly with voltage. Figure 33 shows the average increase in execution time for 25 Mibench applications for CC with voltage frequency scaling (VFS) and MC^2 with Vdd scaling. Figure 34 shows the reduction in average energy-delay product for these two architectures. It is evident that the performance penalty for VFS is very high (about 45% at lowest Vdd) while the performance penalty for MC^2 is only modest (at about 3.5%), as shown in Section 6.2. Thus we find that, for a wide range of voltage, MC^2 consumes significantly lower energy than CC using VFS.

6.7. Discussion

The MC^2 architecture exploits redundancy for fault tolerance and thus could be viewed as an error-correcting mechanism embedded in the cache array. For a wide range of embedded applications, we have shown that MC^2 incurs only modest loss in performance, with significant reduction in energy and with only negligible area overhead. Under equal area constraints, MC^2 provides significantly higher error tolerance than SECDED ECC, leading to higher reduction in Vdd and energy.

Furthermore, we note that MC^2 is a complementary technique that can be used with any existing cache architecture. For instance, it can be combined with SECDED to further increase the error tolerance, leading to even lower Vdd subject to device limitations. MC^2 can also potentially be combined with error-map-based error correction techniques or with redundant rows/columns to increase the performance, for performance-critical applications.

The extent of energy/EDP reduction by use of the MC^2 architecture depends on three factors: (a) the SRAM cell and its inherent probability of failure at the given frequency; (b) target yield; and (c) memory hierarchy (on-chip versus off-chip next level of memory)

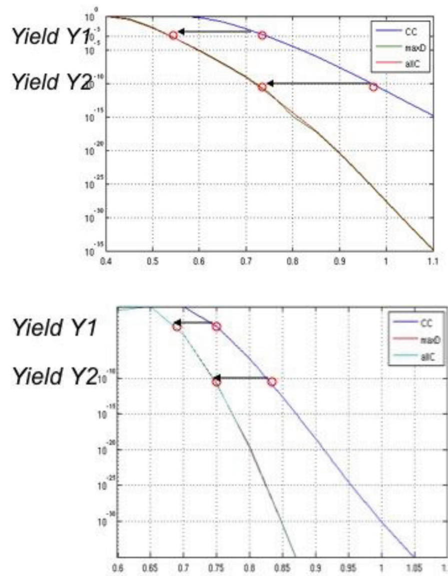


Fig. 35. Improvement in reliability of the cache for two different SRAM cells.

Table IV. IPC Degradation and Reduction in Energy for Two Different SRAM Cells at Different Yield Points

	Y1	***Y*2**
SRAM1	Perf loss = 3% E redn = 25%	Perf loss = 3% E redn = 45%
SRAM2	Perf loss = 3% E redn = 5%	Perf loss = 3% E redn = 20%

and cache miss energy. Figure 35 shows the failure probability of CC and MC^2 for two different SRAM cells. It clearly shows that the reduction in minimum Vdd using MC^2 relative to CC depends on the target yield. In Table IV, we show the performance loss and energy reduction for MC^2 in comparison with CC for these two SRAM cells for two different yields using NME of 5. As NME is increased, the reduction in energy will reduce further, as shown in Section 6.4. Though performance loss remains similar for different SRAM cells and yield points, the reduction in energy is clearly dependent on the SRAM cell used and the yield point. Therefore, for a given SRAM cell and for a given yield, the MC^2 architecture may be very useful, while the same may not be true for another SRAM cell for either the same or different yield.

Multicopy cache can be treated as a superset of single-copy cache. MC^2 can be used as a conventional single-copy cache with a minor change in hardware. By assuming all the data as clean, the MC^2 data correction logic, as described in Section 3, would essentially work as a multiplexer in a CC, outputting the single copy of the data accessed. Therefore a multicopy cache can be used in two modes: (a) a single-copy conventional mode with a single copy per data, and (b) a multicopy mode with multiple copies per data. In the single-copy mode, the cache runs at high Vdd with higher energy consumption and higher performance. By contrast, in multicopy mode the cache runs at lower Vdd with lower energy consumption and lower performance. Furthermore, in

multicopy mode, as Vdd is scaled down from nominal Vdd to Vccmin, the performance and energy consumption are also scaled down. Finally, in multicopy mode, the actual Vdd of operation may be chosen based on the acceptable loss in performance and desired energy consumption.

In all, we note that MC^2 is a cache architecture effective against all kinds of SRAM failures, including soft errors and hard failures. While in this article we have studied the RDT policy for MC^2, many other policies can be examined to yield different levels of power savings and overheads in delay/area.

7. CONCLUSION

In this work, we presented MC^2: a novel cache architecture that allows low Vdd operation for energy savings without affecting the reliability of the system. MC^2 maintains multiple copies of each data item, exploiting the fact that many embedded applications have unused cache space resulting from small working-set sizes. On every cache access, MC^2 detects and corrects errors using these multiple copies. We have shown the MC^2 architecture is efficient, incurring negligible area overhead and only modest performance penalty (<3.5%), but achieving significant energy savings for embedded applications. We have also shown that MC^2 exhibits high levels of error tolerance; thus, we can exploit aggressive voltage scaling for high reductions in energy consumption and energy-delay product for on-chip caches. Our experiments on embedded benchmarks demonstrate that MC^2 reduces total energy consumption by up to 60% over conventional caches. Future work involves further refinement of the RDT policy, developing more efficient reliability policies for MC^2, as well as integrating the MC^2 architecture with other error-tolerant cache architectures using static error maps.

REFERENCES

N. Aboughazaleh, A. Ferreira, C. Rusu, R. Xu, F. Liberato, et al. 2007. Integrated cpu and l2 cache voltage scaling using machine learning. In *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'07).* 41–50.

A. Agarwal, B. Paul, H. Mahmoodi, A. Datta, and K. Roy. 2005. A process-tolerant cache architecture for improved yield in nanoscale technologies. *IEEE Trans. VLSI Syst.* 13, 1, 27–38.

ARM. 2010. ARM cortex-a8 technical reference manual. http://www.arm.com/products/CPUs/ARM_Cortex-A8.html.

T. Austin, E. Larson, and D. Ernst. 2002. SimpleScalar: An infrastructure for computer system modeling. *IEEE J. Comput.* 35, 2, 59–67.

F. Behmann. 2009. Embedded.com - The itrs process roadmap and nextgen embedded multicore soc design. http://www.embedded.com/design/mcus-processors-and-socs/4008253/The-ITRS-process-roadmap-and-nextgen-embedded-multicore-SoC-design.

Y. Cai, M. T. Schmitz, A. Ejlali, B. M. Al-Hashimi, and S. M. Reddy. 2006. Cache size selection for performance, energy and reliability of time-constrained systems. In *Proceedings of the Asia and South Pacific Conference on Design Automation (ASP-DAC'06).*

B. Calhoun and A. Chandrakasan. 2006. A 256kb sub-threshold sram in 65nm cmos. In *IEEE International Solid State Circuits Conference Digest of Technical Papers (ISSCC'06).* 2592–2601.

V. Chandra and R. Aitken. 2009. Impact of voltage scaling on nanoscale sram reliability. In *Proceedings of the Design, Automation, and Test in Europe Conference (DATE'09).* 387–392.

L. Chang, D. Fried, J. Hergenrother, J. W. Sleight, R. H. Dennard, et al. 2005. Stable sram cell design for the 32 nm node and beyond. In *Proceedings of the Symposium on VLSI Technology Digest of Technical Papers.* 128–129.

Q. Chen, H. Mahmoodi, S. Bhunia, and K. Roy. 2005. Modeling and testing of sram for new failure mechanisms due to process variations in nanoscale cmos. In *Proceedings of the 23rd IEEE Symposium on VLSI Test (VTS'05).* 292–297.

Z. Chishti, A. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu. 2009. Improving cache lifetime reliability at ultra-low voltages. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'09).* 89–99.

A. Diril, Y. S. Dhillon, A. Chatterjee, and A. D. Singh. 2005. Level-shifter free design of low power dual supply voltage cmos circuits using dual threshold voltages. *IEEE Trans. VLSI Syst.* 13, 9, 1103–1107.

A. K. Djahromi, A. M. Eltawil, F. J. Kurdahi, and R. Kanj. 2007. Cross layer error exploitation for aggressive voltage scaling. In *Proceedings of the 8th International Symposium on Quality Electronic Design (ISQED'07)*. 192–197.

J. Fritts and W. Wolf. 2000. Multi-level cache hierarchy evaluation for programmable media processors. In *Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS'00)*. 228–237.

J. Fritts, W. Wolf, and B. Liu. 1999. Understanding multimedia application characteristics for designing programmable media processors. In *Proceedings of the SPIE Conference on Media Processors.* Vol. 3655.

P. Genua. 2004. A cache primer. http://www.csd.uwo.ca/~moreno/CS433-CS9624/Resources/AN2663.pdf.

M. Guthaus, J. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the IEEE International Workshop on Workload Characterization (WWC'01)*. 3–14.

M. Y. Hsiao. 1970. A class of optimal minimum odd-weight-column sec-ded codes. *IBM J. Res. Develop.* 14, 4, 395–401.

M. Huang, J. Renau, S.-M. Yoo, and J. Torrellas. 2001. L1 data cache decomposition for energy efficiency. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'01)*. 10–15.

ITRS. 2008. International technology roadmap for semiconductors. http://www.itrs.net/Links/2008ITRS/home 2008.htm.

A. Khajeh, A. Gupta, N. Dutt, F. Kurdahi, A. Eltawil, K. Khouri, and M. Abadir. 2009. TRAM: A tool for temperature and reliability aware memory design. In *Proceedings of the Design, Automation, and Test in Europe Conference (DATE'09)*. 340–345.

J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe. 2007. Multi-bit error tolerant caches using two-dimensional error coding. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'07)*. 197–209.

J. kulkarni, K. Kim, and K. Roy. 2007. A 160 mv robust schmitt trigger based subthreshold sram. *IEEE J. Solid State Circ.* 42, 10, 2303–2313.

S. Lin and D. J. Costello. 1983. *Error Control Coding: Fundamentals and Applications*. Prentice Hall.

M. Makhzan, A. Khajeh, A. Eltawil, and F. Kurdahi. 2007. Limits on voltage scaling for caches utilizing fault tolerant techniques. In *Proceedings of the International Conference on Computer Design (ICCD'07)*. 488–495.

M. Mamidipaka and N. Dutt. 2004. eCACTI: An enhanced power estimation model for on-chip caches. Tech. rep. R-04-28, CECS, University of California, Irvine. http://ftp.cecs.uci.edu/technical_report/TR04-28.pdf.

P. Mazumder. 1993. Design of a fault-tolerant three-dimensional dynamic random-access memory with on-chip error-correcting circuit. *IEEE Trans. Comput.* 42, 12, 1452–1468.

M. Meterelliyoz, J. P. Kulkarni, and K. Roy. 2008. Thermal analysis of 8-t sram for nano-scaled technologies. In *Proceedings of the 13th International Symposium on Low Power Electronics and Design (ISLPED'08)*. 123–128.

S. Mukhopadhyay, H. Mahmoodi, and K. Roy. 2005. Modeling of failure probability and statistical design of sram array for yield enhancement in nanoscaled cmos. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst.* 24, 12, 1859–1880.

R. Naseer and J. Draper. 2008. Parallel double error correcting code design to mitigate multi-bit upsets in srams. In *Proceedings of the 34th European Solid State Circuits Conference (ESSCIRC'08)*. 222–225.

PTM. 2010. Predictive technology model (ptm). http://ptm.asu.edu.

S. Roy. 2009. H-Nmru: A low area, high performance cache replacement policy for embedded processors. In *Proceedings of the 22nd International Conference on VLSI Design.* 553–558.

A. Sasan, H. Homayoun, A. Eltawil, and F. Kurdahi. 2009a. A fault tolerant cache architecture for sub 500mv operation: Resizable data composer cache (rdc-cache). In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'09)*. 251–260.

A. Sasan, H. Homayoun, A. Eltawil, and F. Kurdahi. 2009b. Process variation aware sram/cache for aggressive voltage-frequency scaling. In *Proceedings of the Design, Automation, and Test in Europe Conference (DATE'09)*. 911–916.

S. Schuster. 1978. Multiple word/bit line redundancy for semiconductor memories. *IEEE J. Solid State Circ.* 13, 5, 698–703.

P. Shirvani and E. McCluskey. 1999. PADded cache: A new fault-tolerance technique for cache memories. In *Proceedings of the 17th IEEE VLSI Test Symposium (VTS'99)*. 440.

G. Sohi. 1989. Cache memory organization to enhance the yield of high performance vlsi processors. *IEEE Trans. Comput.* 38, 4, 484–492.

D. Tarjan, S. Thoziyoor, and N. P. Jouppi. 2006. CACTI 4.0. Tech. rep. 2006-86, HP Laboratories. http://www.hpl.hp.com/techreports/2006/HPL-2006-86.pdf.

C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu. 2008. Trading off cache capacity for reliability to enable low voltage operation. In *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA'08)*. 203–214.

W. Wong, C.-K. Koh, Y. Chen, and H. Li. 2007. VOSCH: Voltage scaled cache hierarchies. In *Proceedings of the 25th Conference on Computer Design (ICCD'07)*. 496–503.

C. Zhang, F. Vahid, and W. Najjar. 2005. A highly configurable cache for low energy embedded systems. *ACM Trans. Embed. Comput. Syst.* 4, 2, 363–387.

W. Zhang, S. Gurumurthi, M. Kandemir, and A. Sivasubramaniam. 2003. ICR: In-cache replication for enhancing data cache reliability. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'03)*. 293–300.

W. Zhao and Y. Cao. 2007. Predictive technology model for nano-cmos design exploration. *ACM J. Emerg. Technol. Comput. Syst.* 3, 1.

Received July 2011; revised April 2012; accepted July 2012