C1C: A Configurable, Compiler-Guided STT-RAM L1 Cache

YONG LI, YAOJUN ZHANG, HAI LI, YIRAN CHEN, and ALEX K. JONES, University of Pittsburgh

Spin-Transfer Torque RAM (STT-RAM), a promising alternative to SRAM for reducing leakage power consumption, has been widely studied to mitigate the impact of its asymmetrically long write latency. Recently, STT-RAM has been proposed for L1 caches by relaxing the data retention time to improve write performance and dynamic energy. However, as the technology scales down from 65nm to 22nm, the performance of the read operation scales poorly due to reduced sense margins and sense amplifier delays. In this article, we leverage a dual-mode STT memory cell to design a configurable L1 cache architecture termed C1C to mitigate read performance barriers with technology scaling. Guided by application access characteristics discovered through novel compiler analyses, the proposed cache adaptively switches between a high performance and a low-power access mode. Our evaluation demonstrates that the proposed cache with compiler guidance outperforms a state-of-the-art STT-RAM cache design by 9% with high dynamic energy efficiency, leading to significant performance/watt improvements over several competing approaches.

Categories and Subject Descriptors: [**CCS Concepts**]: Computer Systems Organization—*Multicore architectures*

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: STT-RAM, configurable, compiler, cache, differential

ACM Reference Format:
Li, Y., Zhang, Y., Li, H., Chen, Y., and Jones, A. K. 2013. C1C: A configurable, compiler-guided STT-RAM L1 cache. ACM Trans. Architec. Code Optim. 10, 4, Article 52 (December 2013), 22 pages.
DOI: http://dx.doi.org/10.1145/2555289.2555308

1. INTRODUCTION

Spin-Transfer Torque RAM (STT-RAM) has been proposed for use in Chip MultiProcessor (CMP) cache hierarchies as a potential replacement for SRAM particularly for Last-Level Cache (LLC). STT-RAM caches can leverage near-SRAM performance for read accesses, nonvolatility for reduced leakage power, and increased density and capacity over SRAM. Previous conventional wisdom for STT-RAM is that writes are slower and require more power than their conventional SRAM counterparts. Thus, write performance has been considered the fundamental performance bottleneck and has received the focus of attention for optimization. Several architectural solutions such as hybrid caches with fast and slow writing memory components [Wu et al. 2009; Li et al. 2012]; various methods for preempting, avoiding, and bypassing writes [Zhou et al. 2009; Guo et al. 2010; Qureshi et al. 2010]; and leveraging the asymmetry of writing different logic values [Qureshi et al. 2012] have been proposed to mitigate the write performance problem.

This work is supported by the National Science Foundation, under grant CCF-0702452.
Authors' addresses: Y. Li, Y. Zhang, H. Li, Y. Chen, and A. K. Jones, Department of ECE, University of Pittsburgh, Benedum Hall, 3700 O'Hara Street, University of Pittsburgh, Pittsburgh, PA 15261, USA.

DOI: http://dx.doi.org/10.1145/2555289.2555308

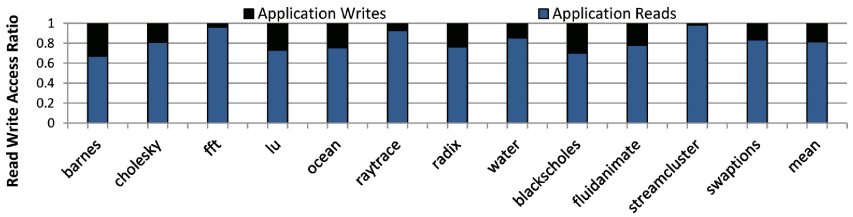

Fig. 1. Application reads versus writes.

A large body of related research, such as those mentioned previously, propose hybrid STT-RAM caches due to the complementary characteristics of STT-RAM versus SRAM in an effort to mitigate challenges that prevent the building of an STT-RAM L1 such as the increased write latency and high dynamic power. Unfortunately, hybrid solutions are less desirable for caches due to additional complexity added to the design. Thus, from the perspective of manufacturing, it is preferable to build caches with homogeneous memory technologies to reduce the integration, verification, and testing costs. To address this problem, two recent techniques to improve the inherent write performance of STT-RAM by tolerating a reduced data retention time [Smullen IV et al. 2011] have led to proposals for use of STT-RAM at the L1 level [Sun et al. 2011] where data access speed is crucial. This makes feasible an *all STT-RAM cache hierarchy* for use in CMPs.

Unfortunately, physical effects of technology scaling down to 45nm and below, in particular process variation, introduce potentially alarming trends in **read performance** of STT-RAM due to reduced sensing margins [Zhang et al. 2012b], especially at the L1 level. Our evaluation of this trend reveals that the read performance problem creates a severe bottleneck for application data reads at higher levels of the cache hierarchy, which typically dominate an application's overall data accesses. Figure 1 shows that for various multithreaded benchmarks, the reads contribute an average of 80% of all the data accesses. A similar trend exists for single-threaded benchmarks. For example, approximately three out of four accesses are reads in SPEC2006 [Ould-Ahmed-Vall et al. 2008].

The goal of this work is to preserve an efficient all-STT-RAM cache hierarchy to avoid significant design and fabrication overheads. A compiler-guided method is presented to leverage a dual-mode STT-RAM cell structure, which utilizes differential sensing to mitigate the increased sense delays and, consequently, the degraded read performance incurred from the technology scaling in L1 caches. In our proposal, accelerated read speed is achieved while maintaining a comparable write performance reported by the state-of-the-art research [Smullen et al. 2011; Sun et al. 2011] that proposes STT-RAM L1 enhancements.

The dual-mode STT-RAM structure allows the construction of L1 cache memories in which a cache block can be configured as a Standard Block (SB) or through differential sensing configured as a Fast read Block (FB) at the expense of higher dynamic write power and reduced capacity. The proposed compiler techniques analyze cache read/write accesses and configure memory cells into the appropriate mode to accelerate data reads without incurring excessive write power. In particular, the *Consecutively Read Blocks* (CRBs) can be identified by the compiler, and mode switching instructions can be inserted to configure the corresponding cache blocks as FB to accelerate read operations efficiently. In contrast, transient read/write access patterns are serviced in SB to retain low power and high density. The proposed compiler techniques avoid the need for expensive runtime detection techniques in hardware, which can add significant complexity and power consumption to the system.

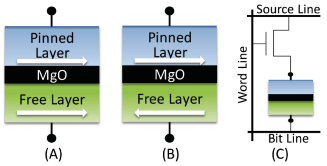

Fig. 2. Illustration of an MTJ and STT-RAM cell.

We make the following contributions compared with prior work:

—We identify and demonstrate that with technology scaling, the read performance is an important new bottleneck that could prevent STT-RAM from being used in timing-sensitive components (e.g., L1 caches) in future CMPs.
—We demonstrate a low-overhead Configurable L1 Cache (C1C) architecture in which a data block can dynamically switch between two modes without data loss. The SB has high density and consumes standard dynamic write power but suffers from a lower read access speed at scaled technology nodes compared to SRAM. The FB offers higher read performance with half the density and double the write power compared to the SB.
—We present several novel compiler analyses, such as Consecutive Temporal Read reuse (CTR), Consecutive Spatial Read reuse (CSR), and Read-Write Distance (RWD). These compiler analyses complement the traditional compiler reuse analysis to handle new challenges.
—The proposed compiler techniques are combined with the configurable cache structure to maximize the read accesses at L1 in FB mode while maintaining a low dynamic write power.

We demonstrate that the proposed C1C brings a 5% performance gain over SRAM and 10% performance improvement with less than 2% dynamic power increase over STT-RAM designs without read optimizations in 22nm technology. In addition, C1C performs closely to (e.g., within 1.5% of) a static all-differential-mode STT-RAM L1 cache with a 26% energy consumption savings. The remainder of the article is organized as follows: Section 2 introduces the state-of-the-art STT-RAM optimizations for write operations as well as the read optimization using differential sensing. Section 3 presents the configurable cache architecture to accelerate data read accesses at the L1 cache level. Additionally, the compiler techniques are introduced to analyze application read/write characteristics that may benefit the configurable cache. We evaluate the performance and power impact of the proposed techniques in Section 4. Section 5 lists related efforts. Finally, we draw conclusions in Section 6.

2. STT-RAM TECHNOLOGY TRENDS AND DESIGN

The building block of STT-RAM is the Magnetic Tunnel Junction (MTJ), which contains two synthetic ferromagnetic layers (pinned and free layer) and one MgO-based tunnel barrier layer [Wen et al. 2012; Zhang et al. 2012a], as illustrated in Figure 2. The magnetic direction of the pinned layer is fixed, while the magnetic direction of the free layer can vary through the application of an external electromagnetic field or spin-polarized current through that layer. When the magnetization directions of the two ferromagnetic layers are parallel, the MTJ is in its low-resistance state (Figure 2(A)). In contrast, when the directions of the two layers are antiparallel, the MTJ resistance is high (Figure 2(B)). The low and high MTJ resistances can be used to represent logic values. In a typical "1T1J" [Hosomi et al. 2005b] STT-RAM cell illustrated in Figure 2(C), one MTJ is connected with one NMOS transistor, which serves as the access controller. This NMOS transistor is typically 1.5 times the size of each of the six

transistors that make up an SRAM cell, leading to the four times density improvement assumed in SRAM replacements with STT-RAM [Sun et al. 2009].

2.1. Write Optimizations

Writes to the MTJ are completed by applying the write current to the cell for a sufficient duration to set the magnetic direction of the free layer to a particular direction. This action is called a "write pulse." When the write pulse width is longer than 10ns, the relationship between write current (I_c) and write pulse width (τ) can be expressed by Equation (1) [Hosomi et al. 2005a]. I_{c0} and τ_0 are the critical write current and the write pulse width, respectively, at $0K$; Δ is the magnetization stability energy barrier, which determines the data retention performance of STT-RAM cells, that is, $T_{retain} \propto e^{\Delta}$ [Diao et al. 2007]. Δ is defined in Equation (2), where K_u is the uniaxial anisotropy energy; V is the volume of a ferromagnetic layer (free layer) of the MTJ; K_B is the Boltzmann constant; and T is the working temperature.

$$I_c(\tau) = I_{c0}\left(1 - \frac{1}{\Delta}\ln\left(\frac{\tau}{\tau_0}\right)\right) \tag{1}$$

$$\Delta = \left(\frac{K_u V}{k_B T}\right) \tag{2}$$

Based on these MTJ design parameters, there are several ways to improve writability [Li et al. 2011a, 2011b; Smullen IV et al. 2011; Sun et al. 2011]. For example, changing the saturation magnetization, the effective anisotropy field, or the thickness of the free layer can lower Δ. In particular, a significantly faster write speed can be achieved at the expense of reduced data retention time, and this technique has been demonstrated to enable L1 STT-RAM caches [Sun et al. 2011]. Thus, we adopt this technique as a baseline to eliminate the impact of excessive write latency and treat this as the state of the art for STT-RAM L1 caches.

2.2. Read Optimization Using Differential Sensing

Reads are completed by sensing the voltage differential in the two resistance states using a read current (I_r) applied for a particular duration, called a read pulse. For all reads to MTJs, there is a probability of disturbing the stored value (Pr_{dis}) that can be expressed as [Hosomi et al. 2005a; Chen et al. 2010]:

$$Pr_{dis} = 1 - exp\left\{-\frac{t}{\tau}exp\left[-\Delta\left(1 - \frac{I_r}{I_c}\right)\right]\right\}. \tag{3}$$

Here, t is the read pulse width. Equation (3) shows that Pr_{dis} is mainly determined by the I_r/I_c ratio. STT design usually uses a global read driver to control I_r and supplies a reference voltage (V_{ref}) to differentiate the high- and low-resistance states of a memory cell. The sense margin of the memory cell thereby is proportional to $I_r \cdot \Delta R/2$, where ΔR is the difference between the high- and the low-resistance states. Improving memory access speed by leveraging differential circuit design is demonstrated to be feasible by several prior efforts [Scheuerlein et al. 2000; Kalter et al. 1990].

As the technology scales, the energy required for writing ($I_c(\tau)$) decreases. To avoid increasing Pr_{dis}, I_r/I_c must remain balanced, requiring proportional reductions to I_r, which in turn reduces the sensing margin ($I_r \cdot \Delta R/2$). Thus, the typical assumption of a 100mV sensing margin required to match the read performance of SRAM should be scaled to more realistic values such as 80mV at 45nm and 40mV at 22nm.

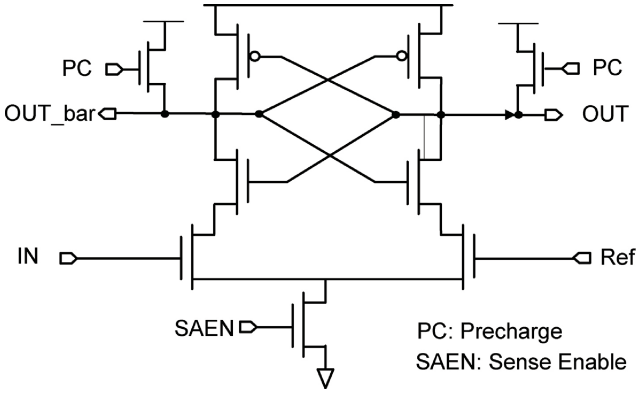

Fig. 3. Sense amplifier design.

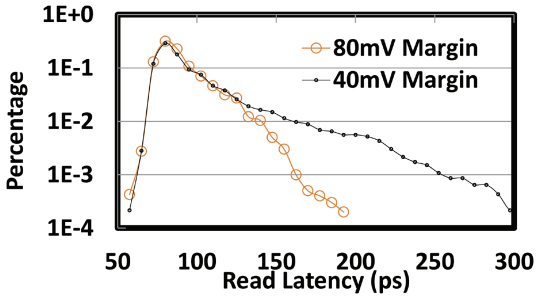

Fig. 4. Voltage mode sense speed distribution with process variation at 22nm.

Sense performance is also heavily impacted by process variation, creating a distribution of sensing times. Sense delays such as 50ps reported in the literature [Smullen IV et al. 2011] often assume the typical case (i.e., the peak of the sense delay curve) and appear quite optimistic, implying no performance difference between the sense margins. To demonstrate this problem, we conducted Monte-Carlo simulations assuming a 5% variation in both transistor size and MTJ shape with an additional 2% intradie variation introduced to account for spatial correlation [Li et al. 2008] of a popular sense-amplifier design in STT-RAM arrays [Cheng et al. 2010] shown in Figure 3.

Figure 4 shows the sensing time distribution curves with 40mV and 80mV margins for conventional and differential sensing, respectively. Based on our modeling, the average sensing times at 40mV and 80mV are 89ps and 99ps, respectively. Therefore, using the average-case sensing latency in our system results in only 99ps − 89ps = 10ps difference between the 40mV and 80mV margins, which would have a minimal impact on the system. Unfortunately, the average sensing delay does not account for a highly reliable read operation, even under variation. In particular, the latency for the sensing amplifier should cover a high percentage ($\geq$99.9%) of the sensing time distribution due to variation in order to ensure that read operations are appropriately reliable. Thus, the performance comparison of sensing requires use of the worse-case sensing time rather than the average time.

To ensure a 99.9% sensing accuracy, we conducted 10,000 Monte Carlo experiments to simulate the sense amplifier under process variation. Based on our modeling, the worst-case sensing delay is around 270ps at a 40mV margin and 170ps at a 80mV margin. We use the worst-case sensing delay in our architecture design to guarantee

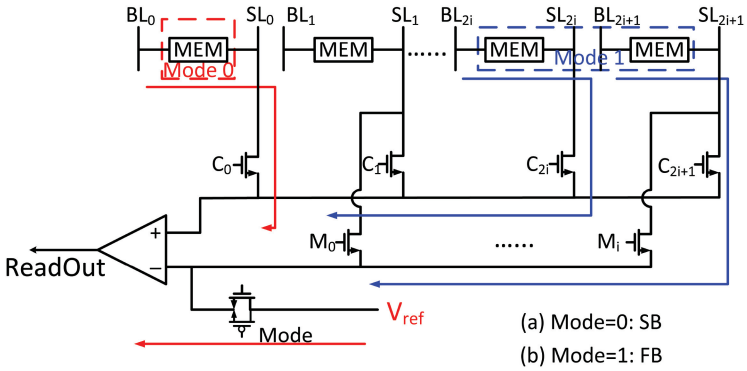

Fig. 5. Configurable SB/FB memory circuit.

Table I. Peripheral Circuitry and Read Latency for Two L1 Cache Examples at 22nm Technology

Cache Config	32K 4-Way			64K 4-Way		
Mem Type/Mode	SRAM	STT SB	STT FB	SRAM	STT SB	STT FB
H-Tree (ns)	0.0338	0.0329	0.0329	0.0378	0.0343	0.0343
Dec+Wordline (ns)	0.1523	0.1343	0.1343	0.1698	0.1406	0.1406
Bitline (ns)	0.096	0.0648	0.0648	0.1643	0.1037	0.1037
SenseAmp (ns)	0.116	0.270	0.170	0.116	0.270	0.170
Total (ns)	0.3981	0.502	0.402	0.4879	0.5486	0.4486

system reliability. In some cases, it may be necessary to ensure a higher reliability (e.g., $\geq$99.99% of sensing time coverage). Unfortunately, to simulate error rates with such high precision requires orders-of-magnitude longer simulation time, becoming intractable. While it may be possible to improve simulation speed and conduct more simulations using a statistical approach, such as the work proposed by Singhee and Rutenbar [2009], this level of device study is beyond the scope of this study and for the remainder of this article we proceed using the ensured 99.9% read accuracy.

In situations where read performance is critical, it is possible to use differential sensing by storing both the value and its complement in adjacent cells in order to double the sense margin ($I_r \cdot \Delta R$) at the expense of reducing storage capacity. As sensing speed is a nonlinear function of sense margin, this increase can provide significant improvements in read performance. A reconfigurable circuit (Figure 5) can be configured into a Standard high-density Block (SB) by comparing the selected cell with V_{ref} or a Fast access Block (FB) by sensing the voltage difference between adjacent cells. To accomplish this, as shown in Figure 5, the mode selection bit `Mode` can be integrated into the source line selection logic to avoid any additional latency in the critical path. The red operation demonstrates an SB read where cell 0 is compared against a threshold V_{ref}. The blue operation indicates an FB read where cell $2i$ is compared against cell $2i + 1$.

As we move to the 22nm technology node, we have created STT-RAM device models and conducted SPICE simulations of our circuit design to determine the performance and power consumption of individual cells. At 22nm, we assume that the sense margins of the sense amplifier in SB and FB are 40mV and 80mV, respectively. To study the architectural impact of the proposed circuit design, we compare the latencies of SRAM with those of STT-RAM in SB and FB for two potential L1 cache configurations, as shown in Table I.[1] A scaled version of CACTI [Shivakumar and Jouppi 2001] is used

[1]As the STT-RAM caches are configurable between SB/FB but two lines are required for FB mode, the capacity reported in the table assumes all blocks are in SB mode.

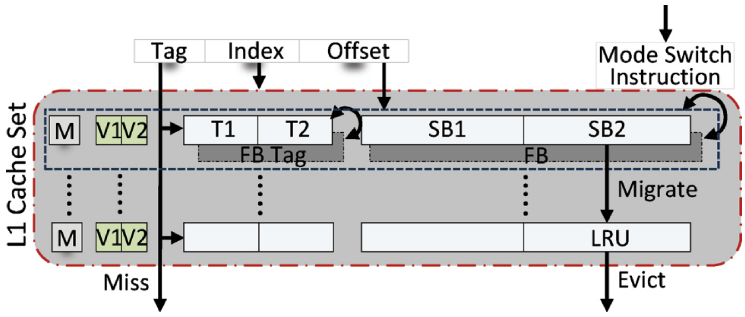

Fig. 6. Configurable L1 cache architecture (C1C).

to derive the peripheral circuitry latencies. The sense amplifier was tuned for best possible performance by sizing of the transistors and timing was derived from an HSPICE simulation.

From Table I, we can see the sensing delay makes up a significant portion of the total cache read latency. Although STT-RAM cells are smaller and thus have faster peripheral circuitry than SRAM, for small caches (e.g., L1), this superiority is negligible and cannot offset the negative impact of the larger sense delay. A large sense delay is a prohibitively expensive penalty and would prevent STT-RAM from being used in L1 caches, for which the access speed is extremely important. By adopting FB, the read access latency of STT-RAM once again becomes comparable with SRAM at L1, making it viable in an all-STT-RAM cache hierarchy.

However, for such a configurable cache to be successful, care must be given to the control of the blocks between SB/FB to balance performance, dynamic power consumption, and capacity. In the following sections we describe a configurable cache design and a compiler-guided method to ensure that differential sensing is used effectively.

3. DUAL-MODE CACHE DESIGN WITH COMPILER SUPPORT

Based on the STT-RAM technology trends and the configurable STT-RAM memory cell from Section 2.2, we present a Configurable L1 Cache (C1C) architecture that can be adapted dynamically at the cache block granularity to offer fast read accesses while minimizing dynamic power overheads. The operational mode of the cache lines will be modified based on application needs determined through compiler analysis. The information will be passed from the compiler to the runtime system via mode configuration instructions instrumented into the code by the compiler. The next several subsections describe these elements in detail.

3.1. C1C Architecture

Our proposed C1C design is shown in Figure 6. In the C1C design, two adjacent cache lines are grouped to form one superline that can operate as an FB line. Each of the two cache lines of the superline can also independently operate in standard (SB) mode (i.e., SB1 and SB2). To accomplish this, each superline contains two respective tags (T1 and T2) and valid bits (V1 and V2) to allow independent SB operation. A mode bit (M) indicates whether the line is storing one FB value or two independent SB values.

Standard cache operations for a statically designed STT-RAM cache can be accomplished in the C1C architecture with only minimal modification. For example, a cache lookup starts by comparing the tags. In C1C, the M bit can be accessed in parallel with the tag lookup such that the resulting data access is conducted in the appropriate SB or FB mode. If the cache access is a read and the M bit indicates the target block is FB,

differential sensing will be used to more quickly read out the value stored in the two adjacent SB blocks.

The C1C cache assumes Least Recently Used (LRU) eviction policy. However, the use of two adjacent lines to store a single FB value requires a slight modification to this policy. If a superline wishes to promote an SB line into FB, an eviction is required. First, the LRU line within the set is identified. If this block is the adjacent SB value in the superline, it is evicted. Otherwise, the LRU line from a different superline is evicted and the adjacent SB value is migrated to the LRU value's previous location. Then the promoted SB line's complementary value is written to the adjacent line as depicted in Figure 6. In contrast, when a line is demoted from FB to SB (often from a period of heavy write behavior), the adjacent SB line becomes vacant and can be used to host a new value without requiring an eviction.

3.2. Design Considerations

The mode change operations described in the previous section result in some operational overheads that must be considered. A write on the same FB block requires twice the amount of dynamic power as a standard write operation due to the writing of the value and its complement. Therefore, it is desirable that an FB line services as many reads but as few writes as possible. Additionally, promoting an SB line to an FB line also results in the overhead of two additional writes (in the worst case) and should only be done when there is high confidence that many successive reads will utilize the line.

From our analysis of applications, L1 caches are particularly sensitive to access latency and dynamic power rather than capacity. To build an effective high-performance and low-power L1 cache, nearly all read accesses in L1 must occur in FB to be competitive with the performance of SRAM. FB reads do not require additional dynamic power as the read current is still only injected once for a differential comparison. However, unlike lower levels in the cache, the number of L1 writes is significant—nearly 20% of accesses, on average. While the write performance in FB is not degraded in comparison with SB as both the cell and its complement can be written in parallel, writes to STT-RAM, even with reduced retention times, still require a significantly higher dynamic power than equivalent technology SRAM and about twice as much dynamic power as SB writes. An effective configurable L1 design should maximize the number of writes completed in SB and reads in FB to avoid dynamic power overheads and improve performance, respectively.

According to our study of various benchmarks, even for heavily written data locations, small numbers of writes are typically interspersed with small numbers of reads. During other application phases, the same location is often extremely heavily read. Thus, to avoid excessive dynamic power for complementary writes and mode reconfigurations, a line should be configured into FB only if the subsequent access pattern exhibits a large amount of consecutive reads without frequently interleaved writes. Thus, the optimization criteria to control the mode of the cache superline is to maximize the number of reads using differential mode (FB) while maximizing the number of writes to standard mode (SB) while minimizing the number of mode switches from SB to FB. Thus, only with a high level of confidence that a number of consecutive reads to the same location will occur (e.g., greater than a particular threshold T) should the line be promoted to FB, with an understanding that this promotion results in an energy overhead for writing the complement and potentially relocating the partner line in the set. If a series of writes intercepts the consecutive read region, the line may be switched back to SB to save dynamic power. This allows the observed intermittent reading and writing pattern to predominantly occur in SB, but many successive reads will be serviced in FB.

Sparc V9 Prefetch Instruction

1 1	fcn	opcode	rs	1	offset
31 30 29	25 24	19 18	14 13	12	0

fcn code	function
0	Prefetch for several reads
1	Prefetch for one read
2	Prefetch for several writes
3	Prefetch for one write
4	Prefetch page
5-15	*Reserved*
16-31	*Implementation-dependent*

Fig. 7. Sparc V9 prefetch instruction format.

Thus, the C1C configuration policy relies on an accurate prediction of CRBs in the applications during execution. The cache block mode is controlled by Mode Switch Instructions (MSIs) inserted by the compiler, whose operands represent the target address for the mode configuration. By utilizing compiler analysis to determine the operational mode for lines, it is possible to circumvent considerable extra hardware overheads required by runtime solutions such as per-line counters, comparators, and mode-switch logic while avoiding potential thrashing possible in runtime solutions. The compiler-guided mode switch instruction can be implemented using the unused bits in an existing instruction,[2] thus avoiding modification to the standard instruction set architecture. For example, the Sparc V9 processor's prefetch instruction contains an *fcn* field in its instruction code that is used to implement prefetch variants. As Figure 7 illustrates, each *fcn* value indicates a different operation and the values 5–15 are reserved for future usage. These bits can also be used to implement the mode switch instruction. In the next section, we describe a compiler analysis methodology that enables accurate insertion of these MSIs into the code.

3.3. Compiler Analysis

In order to accurately identify CRBs to enable appropriate mode switching in the C1C, we can leverage source-code-level information exposed by compile time techniques such as data reuse analysis. Data reuse analysis provides confidence that the data element stored in a particular location will be heavily read, indicating it will be a good candidate for caching in the L1. Unfortunately, this analysis is insufficient to determine if this location should use an FB line. Heavily written locations stored in FB mode induce a significant dynamic write energy overhead. Thus, in this section, we develop compiler techniques based on data reuse analysis to determine patterns of Consecutive Reads (CRs) as an indicator of heavily read locations that are candidates for FB promotion. Given that application behavior changes during execution, the same location could exhibit CR behavior in one phase of the application and interleaved read/write behavior during other phases. The compiler analysis indicates where, within the source code, to insert MSIs as described in detail in the following sections.

3.3.1. CR Analysis for Arrays. Since a precondition for promotion of a location in L1 to an FB line is demonstrable heavy reads, we start with a preliminary data reuse analysis as a basis of our CR analysis. Then, given the heavy reuse candidates, we identify those blocks that have CRs.

[2]Several instruction sets now provide extensibility in the ISA to specifically support custom instructions such as the ones proposed in this work. An example is the *undefined instruction* in the ARM ISA.

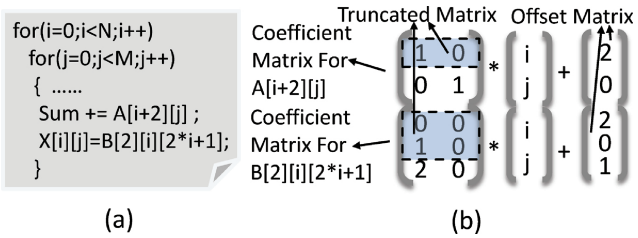

Fig. 8. Array accesses and the corresponding matrix representations: (a) array accesses and (b) matrix representation.

Data Reuse Analysis. Data reuse analysis aims to identify array accesses to the same or nearby locations/elements in nested loops and has been used to detect data dependency and promote locality. One approach to analyze data reuse in nested loops is to represent array accesses as matrix multiplication [Aho et al. 2006]. Consider Figure 8 as an example. Given the array accesses `A[i+2][j]` and `B[2][i][2*i+1]` in the nested loop shown in Figure 8(a), we first convert the subscript functions to the matrix expressions, as illustrated in Figure 8(b). The accessed array elements now can be represented as $C * k + O$, where C is the coefficient matrix, k is the loop index vector, and O denotes the offset vector.

For the array access to have *temporal access reuse*, different loop iterations (i.e., different k) need to reference the same array element (the matrix expression $C * k + O$). Therefore, determining whether the array access has temporal reuse now is equivalent to deriving the condition under which the equation $C * k' + O = C * k'' + O$ has solutions [Wolf 1992] (k' and k'' represent two different index vectors in the iteration space). The necessary and sufficient condition under which this equation has solutions is that C is not fully ranked. In our example, the coefficient matrix of `A[i+2][j]` has a rank of two, indicating no temporal reuse. `B[2][i][2*i+1]` has temporal reuse since the rank of its coefficient matrix is one, which is smaller than its dimension.

A read access exhibits *spatial access reuse* when the innermost enclosing loop index varies only the last coordinate of that array. To discover spatial reuse, we use a truncated coefficient matrix by dropping the last row of the original coefficient matrix, as illustrated in Figure 8(b). If the rightmost column in the truncated coefficient matrix (the coefficients that correspond to the innermost loop index) is a null vector and the rightmost element in the dropped row is nonzero, it is assured that the innermost loop only varies the last coordinate of the corresponding array.

In the aforementioned example, `A[i+2][j]` exhibits spatial reuse because the rightmost column in the truncated matrix (the coefficient corresponding to the innermost loop index j) is a null vector and the rightmost element in the dropped row is nonzero. Using the same rule, we can determine that `B[2][i][2*i+1]` does not have spatial reuse since the innermost loop index j does not vary in the last coordinate of array `B`.

In our analysis, it is necessary to distinguish between read and write accesses within a data block to determine the correct operation mode of the cache (i.e., cache lines with heavy read reuse without heavy write reuse). This distinction between reads and writes is easily extracted from the source as it is used to generate loads and stores in the resulting code. However, in the compiler analysis, it is both the locality of read access and the interaction of reads and writes that must be determined. This is accomplished using CR analysis.

CR Analysis. Given a location with heavy read reuses, it is necessary to determine if these read reuses are consecutive. Read reuses become consecutive if there are

no interleaved writes among the reuses. We discuss how consecutive reads can be identified by adding additional constraints on the basic temporal and spatial read reuse analyses.

First, we define that a data block has *Consecutive Temporal Read reuse* (CTR) if the same address in the block is read multiple times without interleaved writes on the same block. Similarly, a data block is defined to have *Consecutive Spatial Read reuse* (CSR) if nearby addresses within the block are read with no interleaved writes on the same block. A block exhibiting either CTR or CSR is a good candidate for being promoted to FB for read optimizations without causing high dynamic write power. Next we present the compiler analyses for identifying CTR and CSR for array accesses.

CTR Analysis: Identifying CTR is similar to analyzing temporal read reuse except that writes that potentially break CTR should be taken into consideration. A read to an n-dimensional array A within an m-deep loop nest has the form of $\ldots = A[fr_1(L)]\ldots[fr_n(L)]$, where $fr_*(L)$ are the subscript functions for the array read defined on a set of loop indices $L = i_1, \ldots, i_m$, from the outermost to the innermost. The corresponding lower bounds and upper bounds of these indices are $l_1, \ldots, l_m$ and $u_1, \ldots, u_m$, respectively. We now consider a special scenario of temporal read reuse: array accesses that exhibit temporal read reuse over the innermost loop nest. This is the case when the same address is accessed many times because the innermost loop iterates over its index i_m and the reuse distance [Ding and Zhong 2003] is smaller than $u_m - l_m$. We denote this special type of read reuse as *i-reuse* and other temporal reuse (over outer loops) as *o-reuse*. An i-reuse can be identified by examining whether the rightmost column of the coefficient matrix of an array access is all zeros, indicating that the array access is invariant to the innermost loop index. Take Figure 8 as an example; the array access `B[2][i][2*i+1]` exhibits i-reuse because its coefficient matrix has an all-zero rightmost column and thus the same element of `B` is accessed repeatedly as the innermost loop index j iterates.

This definition implies that array reads with i-reuse are temporally close to each other as the same array element is reused multiple times during innermost loop iterations. Thus, the read reuse distance (RRD), essentially the number of loop iterations between read accesses, is small for reads with i-reuse. In other words, it is less common for this type of read reuse to be interleaved by a write and thus i-reuse tends to result in CTR. However, it is still possible for a write, with the form of $A[fw_1(L)]\ldots[fw_n(L)] = \ldots$, where $fw_*(L)$ are the subscript functions for an array write in a similar fashion as described for reads mentioned earlier, to break the CTR pattern if the following conditions are satisfied:

$$\begin{aligned} &fw_1 = fr_1,\ fw_2 = fr_2, \ldots,\ fw_{n-1} = fr_{n-1} \text{ and} \\ &|fw_n - fr_n| \equiv c < T (\text{c is a constant}). \end{aligned} \tag{4}$$

The first condition in Equation (4) ensures that the read and the write index the same locations at all but the last dimension (n_{th}) of the array A. Given the first condition is met, the second condition $|fw_n - fr_n|$ calculates the *read-write distance* (RWD) over the last dimension and examines if this distance is a constant whose absolute value is smaller than T. If the absolute value of RWD is a constant smaller than T, then the write consistently occurs less than T elements on the array away from the read and, thus, is likely to break the CTR on a cache block at runtime. If the RWD is not a constant or larger than T, then there is typically CTR as the write will be far enough away from the consecutive reads, which are sufficiently close together as in the case of i-reuse. For example, consider the array accesses in Figure 9 assuming a value $T = 3$. In Figure 9(a), the value $|fw_n - fr_n|$ is a constant 4 and the analysis would determine that CTR does exist, as illustrated in Figure 10(a). In Figure 10(a), a read on the

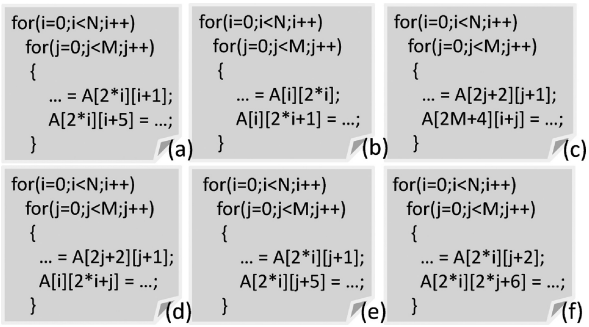

Fig. 9. CTR and CSR code examples.

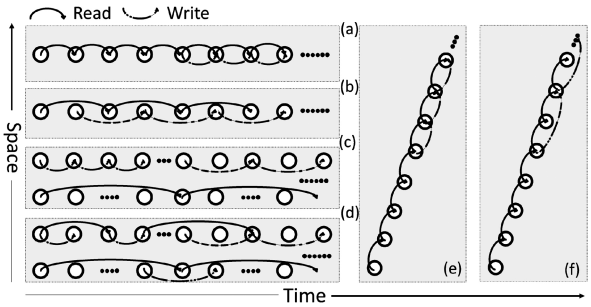

Fig. 10. CTR and CSR access patterns.

data element is always four iterations away from a corresponding write on the same element. In contrast, Figure 9(b) produces a constant $|fw_n - fr_n|$ value of 1, indicating CTR is broken by a nearby write, as illustrated in Figure 10(b).

For o-reuse, the RRD is typically much larger than that of an i-reuse. Thus, several reads with o-reuse are more likely to be interrupted by a write. We determine that an o-reuse is CTR if there is no write to the same array location during that loop (and all contained within it) that overlaps with the read using the following conditions:

$$fw_{1_{max}} < fr_{1_{min}} || fw_{1_{min}} > fr_{1_{max}} || \ldots || fw_{n_{max}} < fr_{n_{min}} || fw_{n_{min}} > fr_{n_{max}}. \quad (5)$$

In Equation (5), $fw_{*_{min}}$, $fw_{*_{max}}$, $fr_{*_{min}}$, and $fr_{*_{max}}$ can be calculated from the lower and upper bounds l_* and u_* of the corresponding loops. The condition in Equation (5) guarantees that the indexed locations of the write are entirely out of the scope of the locations accessed by the read for at least one dimension of the array. An o-reuse-induced CTR example is shown in Figure 9(c), in which the array write `A[2M+4][i+j]` falls out of the range of the array read `A[2j+2][j+1]` for any possible j value within the loop bounds. In Figure 9(d), the read and write could potentially interfere with each other and thus no CTR is detected. The access patterns for the aforementioned two examples are illustrated in Figure 10(c) and Figure 10(d), respectively. In Figure 10(c), the reads occur on one row of the accessed array while the writes occur on an entirely different row, indicating that CTR exists. On the other hand, in Figure 10(d), reads and writes are temporarily interleaved on the same data rows, implying no CTR.

CSR Analysis: Analyzing CSR requires identifying writes that interfere with spatial read reuse. Given an array access with spatial read reuse in a loop nest, CSR exists if there are no writes on the same array in the loop. In the presence of potential

interfering writes, we determine that CSR exists if one of the following two conditions is met (given the same notation used in CTR analysis):

—The read and write do not overlap as can be verified by Equation (5).
—The subscript functions satisfy:

$$fw_1 = fr_1,\ fw_2 = fr_2, \ldots, fw_{n-1} = fr_{n-1} \text{ and } |fw_n - fr_n| > T. \tag{6}$$

In the presence of spatial read reuse, the first condition (Equation (5)) strongly guarantees that no writes overlap with the read reuse and, thus, CSR exists. In the second condition, the term $|fw_n - fr_n| > T$ in Equation (6) ensures that the RWD is large enough that the write is never close enough to the read to break the CSR as the innermost loop iterates. This condition can be verified in the following three cases:

First, when the RWD in the last dimension $|fw_n - fr_n|$ is a constant, its absolute value should be larger than the threshold T:

$$|fw_n - fr_n| \equiv c > T (\text{c is a constant}). \tag{7}$$

If Equation (7) is satisfied, the RWD for the last dimension of the array is consistently large enough that as the loop iterates, the write will not occur until at least a sufficient number of loop iterations occur such that CSR can be ensured. An example of this scenario is presented in Figure 9(e) and Figure 10(e), where the write `A[2*i][j+5]` is four iterations away from the read `A[2*i][j+1]`, which is $>T$ if $T = 3$.

Next, when fw_n is larger than fr_n at the innermost loop lower bound l_m, $|fw_n - fr_n|$ should be a monotonically increasing function on the innermost loop index i_m:

$$fw_n|_{i_m=l_m} - fr_n|_{i_m=l_m} > T \text{ and } (fw_n - fr_n)|_{i_m=x} > (fw_n - fr_n)|_{i_m=y}\ \forall x > y. \tag{8}$$

The first condition in Equation (8) ensures the RWD is larger than T when the innermost loop starts to iterate (index i_m equals the lower bound l_m). The second condition ensures that the RWD keeps increasing as the innermost loop iterates (RWD increases as i_m increases) so that the RWD becomes larger and larger than T, resulting in CSR. An example of this scenario is presented in Figure 9(f) and Figure 10(f). At $j = 0$, the last index of the write is 6 and the read is 2 with a distance of 4, and as j increases, the $2 * j$ factor increases this distance (5 for $j = 1$, 6 for $j = 2$, etc.). Thus, the distance will always be $>T$.

Finally, when fw_n is smaller than fr_n at the innermost loop lower bound l_m, essentially the converse of Equation (8), $|fw_n - fr_n|$ should be a monotonically decreasing function on the innermost loop index i_m as in Equation (9).

$$fw_n|_{i_m=l_m} - fr_n|_{i_m=l_m} < T \text{ and } (fw_n - fr_n)|_{i_m=x} < (fw_n - fr_n)|_{i_m=y}\ \forall x > y \tag{9}$$

3.3.2. CR Analysis for Linked Structures. Because linked data structures are not typically allocated consecutively in memory, determining CR can be reduced to identification of CSR, which is common when several nearby fields in an object are read consecutively. To analyze the CSR for linked data structures such as linked lists and trees, a Control Flow Graph (CFG) of the program is constructed. A CFG $G = (V, E, r)$ is a directed graph, with nodes V, edges E, and an entry node r. Each node v in V is a basic block, which consists of a sequence of statements that have one exact entry point and exit point. To simplify the code structure, a series of traditional compiler optimizations such as expression folding and branch elimination are applied on the CFG. Then the CFG is traversed while the following rules are examined to determine whether a sequence of data reads exhibits CSR:

—The analyzed memory reads are common pointer-based dereferences. That is, these memory reads only differ in their offsets from a common base pointer.

—There are at least T data reads whose offsets fall into a specified address range.[3] This is to guarantee the memory reads are within a sufficiently small scope in the address space.
—There are no function calls or writes within the same block range among the analyzed reads.
—The reads composing the CSR sequence are either in the same basic block or in a set of direct successor basic blocks that meet these criteria. If there are conditionals, the second criterion must be satisfied in all branches.

These rules guarantee that the analyzed reads are mapped to the same memory block and result in consecutive read behavior at runtime. The pseudocode for identifying CSR in basic blocks and their successors is presented in Algorithm 1. It iterates over each basic block and collects relevant information on memory reads (i.e., base pointers and offsets). It organizes the collected information from different phases into a table, where each function call initiates a new phase. At the exit of each basic block, the table is traversed and the corresponding entries are marked to indicate the identified CSR. For unmarked entries in the table, the first phases of all direct successors of the current basic block are further analyzed for potential CSR across basic blocks in the CFG.

ALGORITHM 1: Pseudocode for CSR identification of linked data structures in CFG $G(V, E, r)$

```
begin
    for each basic block b_i ∈ V do
        create table H; phase = 0;
        for each statement s_j ∈ b_i do
            if s_j is function call then
                phase++;
            else
                get the base pointer bp of RHS(s_j); get the offset o of RHS(s_j);
                if ∃ entry Y ∈ H such that bp = Y.bp && phase = Y.phase then
                    append o to Y.r;
                else
                    create a new entry with o, phase and bp and push it into H;
                get the base pointer bp' of LHS(s_j);
                get the offset o' of LHS(s_j);
                if ∃ entry Y ∈ H such that bp' = Y.bp && phase = Y.phase then
                    append o' to Y.w;
        traverse H and mark CSR for each entry Y with more than T − 1 read offsets Y.r in
        a single phase that are within the specified range without an interrupting write
        offset Y.w;
        for each unmarked entry Y' ∈ H do
            for each block b_j ∈ ⋃ SUCC(b_i) do
                search Y'.bp in the first phase in b_j and compute the total number of offsets
                n that are within the specified range;
                if n < T then
                    continue to process next unmarked entry;
            mark entry Y';
end
```

[3]This range depends on the size of the cache block. For example, in a cache with a 64-byte block size, this range is 64 bytes.

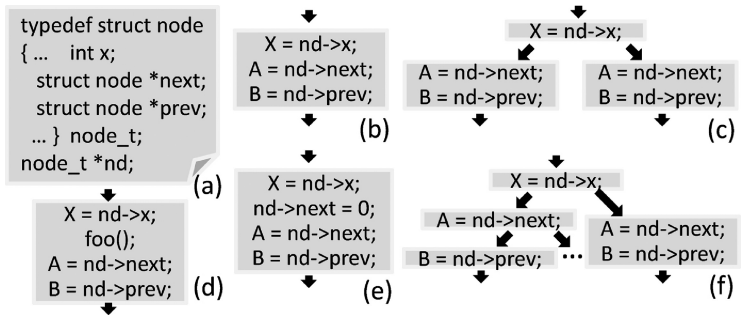

Fig. 11. Code and control flow graph examples for CSR identification (T = 3): (a) type definition code; (b) CSR in the same basic block; (c) CSR across one basic block and all its successors; (d) CSR broken by function call; (e) CSR broken by write; (f) CSR broken by one successor.

Figure 11 provides various cases to explain the CSR identification for linked structures. As defined in Figure 11(a), the pointer *nd* is declared to point to a data structure with the type *node_t*. Since the data members *x*, *next,* and *prev* have integer/pointer type and are adjacent fields in the same data structure, they are consecutive in the address space and, thus, would typically reside in the same memory block such as a cache line. In the case of Figure 11(b), the three memory reads in the same basic block (i.e., *X=nd->x*, *A=nd->next,* and *B=nd->prev*) have the common base pointer *nd* and there are no interleaved function calls or writes. Thus, Figure 11(b) exhibits CSR. The program in Figure 11(c) also has CSR since both successors of the basic block *X=nd->x* lead to CSR. Figures 11(d) and 11(e) do not exhibit CSR due to the presence of the function call *foo()* and write *nd->next=0*. In Figure 11(f), one of the direct successors *A=nd->next* only has one common pointer- based read and, thus, will not be marked as having CSR.

4. EVALUATION

To evaluate the effectiveness of the configurable cache C1C, we use Wind River Simics [Magnusson et al. 2002] to simulate a 16-core CMP with the cache architecture as described in Table IV. We compare the C1C scheme with an SRAM-only design and the leading STT-RAM technique with reduced retention time (26.5μs) and dynamic data refresh in the L1 cache but without differential sensing (STT-RR) [Sun et al. 2011]. All STT-RAM designs use a 64M (4M/core) LLC, while the SRAM cache uses a 16M (1M/core) LLC for a same die area comparison, similar to the architecture evaluated in Sun et al. [2011]. The scheme FL1 is statically configured with all differential blocks (i.e., FB blocks) on top of STT-RR to optimize read performance for L1. C1C is the proposed configurable L1. Just as with STT-RR, both FL1 and C1C include standard STT-RAM at the L2 and L3 cache levels. The mode configuration information is inserted into the source code using source-to-source compilation passes as described in Section 3.3. This information can be inserted into the source using special reserved instructions in the instruction set or extensible instructions with reserved fields from standard instructions (see Section 3.2). In our experiments, the compilation flow inserts a new instruction—simulated using a Simics MAGIC instruction—to notify the runtime system of consecutive read addresses prior to the actual write/read operations. This technique is similar to the mechanism used by existing compiler-instrumented prefetching techniques that insert special instructions [Luk and Mowry 1999]. Our input workloads consist of parallel benchmarks from the SPLASH-2 [Arnold et al. 1992] and PARSEC [Bienia et al. 2008] benchmark suites, as detailed in Table II.

Table II. Benchmarks

Benchmark Suites	Benchmark	Description	Input
SPLASH-2	barnes	Simulation	1048576 particles
	cholesky	Matrix Calculation	tk29.O
	fft	FFT Algorithm	2^26 integers
	lu	Matrix Calculate	2048×2048 matrix
	ocean	Simulation	1026×1026 matrix
	raytrace	Rendering	*teapot* input
	radix	Integer Sort	104857600 radix
	water	Simulation	3375 molecules
PARSEC	blackscholes	Financial Analysis	200000 options
	fluidanimate	Animation	in-35K.fluid
	streamcluster	Data Mining	1024 data points
	swaptions	Financial Analysis	256 swaptions

Table III. Overheads for Ocean with Different *T* Values

T	2	3	4	5	6
Optimized Reads	5.12e+7	5.10e+7	5.08e+7	5.07e+7	5.06e+7
Standard Reads	782107	969789	1136674	1262467	1388100
Optimized Reads (%)	98.50%	98.14%	97.81%	97.57%	97.33%
Complement Writes	187682	166885	125793	125633	125395
Original Writes	1866391	1866391	1866391	1866391	1866391
Write Overhead (%)	10.06%	8.94%	6.74%	6.73%	6.72%

4.1. Effectiveness of the Threshold Analysis

As described in Section 3.3, the compiler analysis determines CRBs by examining use distance threshold T between reads and writes. The determination of a reasonable distance requires some consideration of application characteristics. As we mentioned in Section 3.2, heavily accessed data locations tend to have periods of heavy read access and possibly periods of intermittent reads and writes. We examined this behavior in more detail for a subset of our benchmarks. Table III shows the results in varying T, in effect ensuring that a minimum of T consecutive reads must be present to form a CRB and promote the line to FB. First, we note that even for $T = 2$, the classification mechanism is generally effective and identifies a dominant percentage of the application data reads while only incurring a fairly small number of complementary writes due to mode switches or FB writes. A larger T value guards more consecutive reads for an FB promotion, yielding a configuration with fewer optimized reads but also lower write overhead. By comparing the achieved read optimization and the write cost for four benchmarks, ocean, swaptions, blackscholes, and radix, across different thresholds, a value of $T = 4$ was selected for our final evaluation (Table IV) and it seems to work well for the SPLASH and PARSEC benchmarks.

Given this threshold, Figure 12 reports the percentage of consecutive read regions that can be identified by the proposed compiler analyses (CTR and CSR) compared with all of the CRBs present in a runtime trace. For various applications, from 70% to over 95% and an average of 85% of consecutive reads can be detected by the compiler techniques. This guarantees a high percentage of application data reads to be serviced in fast mode blocks (FBs). We verified this in Figure 13, which shows the number of reads conducted in standard mode (S Reads) and differential mode (F reads) by employing FB mode switching with a threshold of four consecutive reads. In the C1C L1 cache, all the benchmarks have more than 80% of their data reads optimized in FBs. Many benchmarks have over 90% optimized reads, leading to 91% read optimizations

Table IV. Architectural Parameters
The read/write latency for LLC shown in this table is the raw access time excluding the network traversal latency.

Basics	16 cores, 2 issue width, 3.5GHz CPUs			64-bit Solaris 10 OS		4 × 4 mesh network with 3-cycle latency per hop, 4GB main memory, 150-cycle latency	
Caches	Private L1 Cache (MESI)			Private L2 Cache		Shared L3 Cache	
	32K 4-way 64B blk			256K 8-way 64B blk		16M 16-way 64B blk	64M 16-way 64B blk
	SRAM	STT S	STT F	SRAM	STT	SRAM	STT
Size (mm^2/core)	0.048	0.031		0.233	0.085	0.96	1.006
Read Latency (cycles)	2	3	2	4	4	5	6
Write Latency (cycles)	3	5	5	3	26	4	27
Read Energy (nJ)	0.029	0.014	0.014	0.032	0.022	0.054	0.046
Write Energy (nJ)	0.031	0.094	0.188	0.036	0.117	0.06	0.26
Leakage Power (mW)	149	63	63	664	138	1,249	471

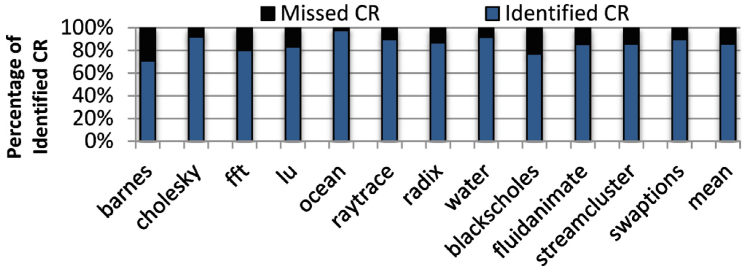

Fig. 12. Percentage of identified consecutive read reuse.

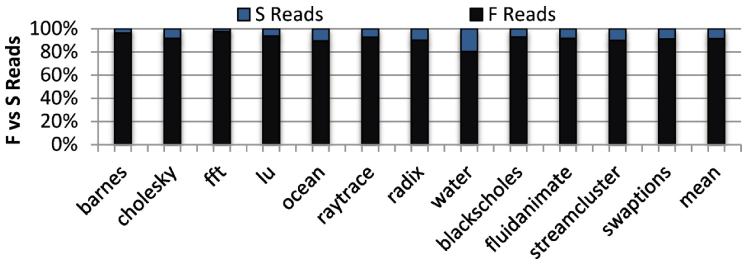

Fig. 13. Reads in different modes (optimized reads).

on average. Since the L1 cache absorbs most of the reads issued by an application, the accelerated reads in C1C are likely to bring a significant performance gain for the entire cache hierarchy.

Due to the effectiveness of the mode configuration control, C1C accelerates a high percentage of application reads while only a tiny fraction of the writes occur in FBs, which incur additional write energy for complementary writes. Figure 14 shows that the percentages of complementary writes (i.e., FB writes) are low for all the tested applications using the same configuration for Figure 13. For many applications such as RAYTRACE and FLUIDANIMATE, the write overheads are almost negligible. On average, there is only 7% of write operations that need complementary writing. This drastically reduces the L1 dynamic power overhead since STT-RAM's write power

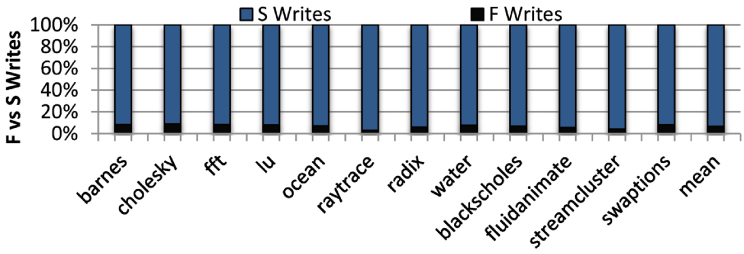

Fig. 14. Writes in different modes (write overhead).

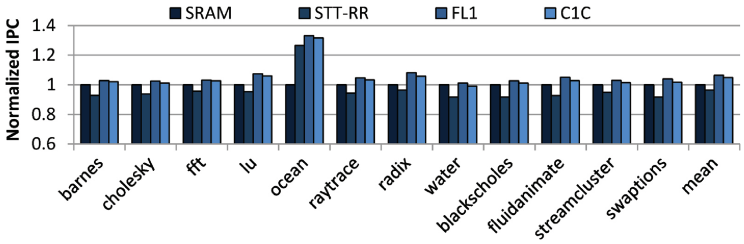

Fig. 15. Performance (IPC) comparison (norm. to SRAM).

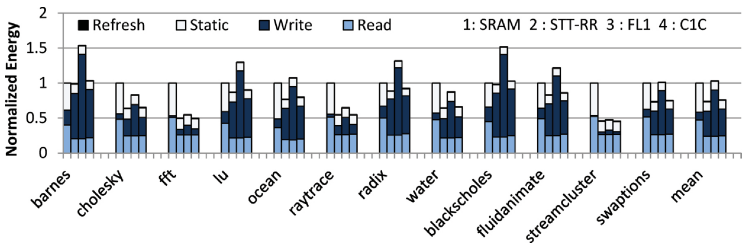

Fig. 16. Energy consumption (norm. to SRAM).

is typically high and a dominating factor of the entire dynamic cache power. The remaining results presented assume use of a mode switch threshold of $T = 4$.

4.2. Performance Evaluation

To conduct performance evaluations, we use the architectural parameters from Table IV that are derived from HSPICE simulation of 22nm STT-RAM sensing times from Table I and scaled versions of CACTI [Shivakumar and Jouppi 2001] for the SRAM and peripheral circuit latencies. The performance comparison, presented as Instructions Per Cycle (IPC) normalized to SRAM, is contained in Figure 15. STT-RR performs poorly compared to SRAM in spite of the capacity advantage due to the high read latency at L1. For all the benchmarks except OCEAN, which is extremely capacity sensitive, STT-RR performs worse than SRAM and leads to an average of 4% performance degradation. In contrast, by servicing all L1 reads with the read optimization technique, FL1 provides over 6% improvement over SRAM and 10% improvement over STT-RR at the expense of double the amount of write power at L1 for complementary writes. The performance of C1C is within 1.5% of FL1 and 5% higher than SRAM and nearly 9% higher than STT-RR.

Figure 16 provides an energy comparison of the relevant schemes normalized to SRAM. STT-RR provides a 27% energy reduction over SRAM on average, and as

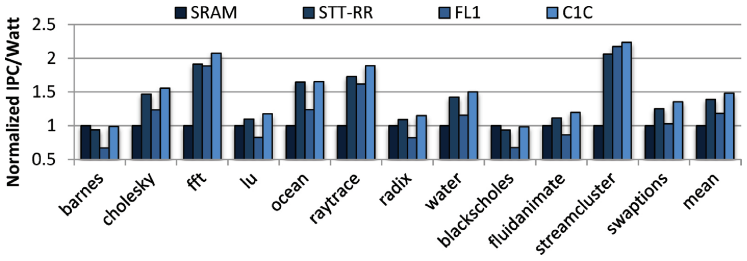

Fig. 17. Performance per watt comparison (norm. to SRAM).

previously demonstrated the refresh energy is negligible [Sun et al. 2011]. As expected, FL1 requires much higher dynamic energy (shown separately as read and write energy in Figure 16) than SRAM and STT-RR due to the fact that all its L1 writes are conducted in FBs, incurring a high dynamic write energy overhead. For benchmarks exhibiting heavily writes such as BARNES, RADIX, and BLACKSCHOLES, the dynamic power increases are unacceptable. In contrast, C1C drastically reduces the dynamic energy compared to FL1 and only requires slightly higher energy than STT-RR since there is still a small portion of writes being serviced in FBs (Figure 14) for the S- to F-mode switches. Compared to FL1, C1C brings similar performance with 26% less total cache energy largely due to the write energy reduction, as can be observed from Figure 16. Because of the leakage and read energy reduction, C1C also brings a 24% total energy savings over SRAM caches.

The overall benefit of C1C is demonstrated in Figure 17, which presents the IPC/watt of various schemes normalized to SRAM. STT-RR suffers from degraded performance but drastically reduced power consumption and thus brings a total of 38% IPC/watt benefit. FL1 enhances performance but increases the dynamic power leading to only 18% IPC/watt gain over the baseline. C1C brings both a performance and energy benefit by dynamically configuring the modes based on application needs that results in the maximum average IPC/watt improvement of 48% compared to the baseline.

5. RELATED WORK

Due to the asymmetric nature of read/write characteristics, intensive research efforts have been made to mitigate the write-induced penalty for various emerging Non-Volatile Memory (NVM) technologies. For STT-RAM, the excessive long write delay can be significantly reduced by relaxing the nonvolatility [Smullen et al. 2011] or reducing data retention time [Sun et al. 2011] with a dynamic data refreshment support to retain datum. The write energy can be also saved by adopting the early write termination [Zhou et al. 2009], which avoids unnecessary writes in STT-RAM cells. There are similar techniques proposed for combating a write-related penalty in Phase-Change Memory (PCM). Qureshi et al. [2012] recently proposed PreSET, a scheme aimed to improve read/write performance by leveraging the asymmetry in writing different logic values. Their earlier effort [Qureshi et al. 2010] attempted to alleviate the penalty of pending reads caused by long write delay using write cancellation and write pausing.

NVMs have also been substantially explored at the architecture level. Guo et al. [2010] use STT-RAM to redesign a number of non-write-intensive micro-architectural components and adopt a subbank write buffering policy with read-write bypassing to increase write throughput and hide the high write latency. Wu et al. [2009] studied the Region-based Hybrid Cache Architecture (RHCA) and Level-based Hybrid Cache Architecture (LHCA) with STT-RAM and PCM. Rasquinha et al. [2010] address the

high write energy of STT-RAM by adopting a new replacement policy that increases the residency of dirty lines at the expense of a higher miss rate.

Other related efforts involve compiler-assisted techniques for cache enhancement. In the work by Lu et al. [2009], a polyhedral model is used to perform localization analysis based on the Farkas Lemma and Fourier-Motzkin algorithms. The goal is to find a data layout transformation to promote locality of accesses. Their work assumes a fixed way of partitioning the iteration space (hence the data space) among parallel threads and works in a similar way to that of some conventional parallelizing compilers.

Li et al. [2012] propose a compiler-assisted technique to improve the performance and energy efficiency for embedded systems with STT-SRAM hybrid caches by reducing the migration overhead. In particular, they identify migration-intensive memory blocks through compiler analysis and give those blocks higher priority to be placed in the SRAM component to avoid frequent migration and long write latency on STT-RAM.

Chen et al. [2012] develop a compiler pass that provides data placement hints to reduce STT-RAM write frequency on a customized hardware that can correct the compiler hints based on runtime cache behavior. Similar to our work, their proposed compiler leverages the concept of memory reuse distance. However, the reuse distance concept used in their work is a classical one and is not aware of the read-write interleaving patterns. Thus, it cannot identify optimization opportunities brought by consecutive reads.

The uniqueness that sets our work apart from prior efforts is that we address the emerging read bottleneck from technology scaling and focus on using novel compiler analysis to improve read performance, rather than tackling the writing problem. We achieve the goal by leveraging new compiler analysis for the read-write access pattern and a reconfigurable design.

6. CONCLUSION

In this article, we identify several read performance bottlenecks in scaled STT-RAM cells. Using a proposed runtime configurable cache design C1C, guided by novel compiler techniques of consecutive read analysis, we *enable an all-STT-RAM cache design* through read optimizations at the L1 cache with performance benefits and energy efficiency. Experimental results demonstrate that C1C provides a better than 9% performance gain over the state-of-the-art all-STT-RAM cache without read optimizations at L1 and a 30% performance-per-watt improvement compared to a nonconfigurable cache with read optimizations. In our future work, we plan to explore runtime alternative approaches for configurable caches enabling read optimizations at different cache levels. Further, a detailed study of refresh times required for relaxed retention STT-RAM caches may be completed and techniques explored to minimize this overhead in the context of C1C caches. Finally, to evaluate the potential impact of the proposed technique on system reliability, we will study sensing errors based on rare event simulation tools [Singhee and Rutenbar 2009].

REFERENCES

AHO, A. V., LAM, M. S., SETHI, R., AND ULLMAN, J. D. 2006. *Compilers: Principles, Techniques, and Tools,* 2nd ed. Addison Wesley.

ARNOLD, J. M., BUELL, D. A., AND DAVIS, E. G. 1992. Splash 2. In *Proceedings of 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'92)*.

BIENIA, C., KUMAR, S., SINGH, J. P., AND LI, K. 2008. *The PARSEC Benchmark Suite: Characterization and Architectural Implications*. Technical Report TR-811-08. Princeton University.

CHEN, Y., LI, H., WANG, X., ZHU, W., XU, W., AND ZHANG, T. 2010. Combined magnetic- and circuit-level enhancements for the nondestructive self-reference scheme of STT-RAM. In *Proceedings of the 2010 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'10)*.

CHEN, Y.-T., CONG, J., HUANG, H., LIU, C., PRABHAKAR, R., AND REINMAN, G. 2012. Static and dynamic co-optimizations for blocks mapping in hybrid caches. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'12)*. ACM, New York, NY, 237–242.

CHENG, C.-T., TSAI, Y.-C., AND CHENG, K.-H. 2010. A high-speed current mode sense amplifier for Spin-Torque Transfer Magnetic Random Access Memory. In *Proceedings of the International Midwest Symposium on Circuits and Systems (MWSCAS'10)*. 181–184.

DIAO, Z., LI, Z., WANG, S., DING, Y., PANCHULA, A., CHEN, E., WANG, L.-C., AND HUAI, Y. 2007. Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory. *Journal of Physics: Condensed Matter 19,* 16, 165209.

DING, C. AND ZHONG, Y. 2003. Predicting whole-program locality through reuse distance analysis. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'03)*. ACM, New York, NY, 245–257.

GUO, X., IPEK, E., AND SOYATA, T. 2010. Resistive computation: Avoiding the power wall with low-leakage, STT-MRAM based computing. In *Proceedings of the International Symposium on Computer Architecture (ISCA'10)*.

HOSOMI, M. AND OTHERS. 2005a. A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-RAM. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM'05)*.

HOSOMI, M., YAMAGISHI, H., YAMAMOTO, T., BESSHA, K. AND OTHERS. 2005b. A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-RAM. *IEDM Technical Digest 2*, 25, 459–462.

KALTER, H. L., STAPPER, C. H., JR. BARTH, J. E., DILORENZO, J., DRAKE, C. E., FIFIELD, J. A., JR. KELLEY, G. A., LEWIS, S. C., VAN DER HOEVEN, W. B., AND YANKOSKY, J. A. 1990. A 50-ns 16-Mb DRAM with a 10-ns data rate and on-chip ECC. *IEEE Journal of Solid-State Circuits 25*, 5, 1118–1128.

LI, H., WANG, X., ONG, Z.-L., ZHANG, Y., WONG, W.-F., WANG, P., AND CHENG, Y. 2011a. Performance, Power and Reliability Tradeoffs of STT-RAM Cell Subjective to Architecture-level Requirement. In *Proceedings of the IEEE International Magnetics Conference (InterMag'11)*.

LI, H., WANG, X., ONG, Z.-L., WONG, W.-F., ZHANG, Y., WANG, P., AND CHEN, Y. 2011b. Performance, power, and reliability tradeoffs of STT-RAM cell subjective to architecture-level requirement. *IEEE Transaction on Magnetics 47,* 10, 2356–2359.

LI, J., AUGUSTINE, C., SALAHUDDIN, S., AND ROY, K. 2008. Modeling of failure probability and statistical design of spin-torque transfer magnetic random access memory (STT MRAM) array for yield enhancement. In *Proceedings of DAC*. ACM, New York, NY, 278–283.

LI, Q., ZHAO, M., XUE, C. J., AND HE, Y. 2012. Compiler-assisted preferred caching for embedded systems with STT-RAM based hybrid cache. *SIGPLAN Not. 47,* 5, 109–118.

LI, Y., CHEN, Y., AND JONES, A. K. 2012. A software approach for combating asymmetries of non-volatile memories. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'12)*.

LU, Q., ALIAS, C., BONDHUGULA, U., HENRETTY, T., KRISHNAMOORTHY, S., RAMANUJAM, J., ROUNTEV, A., SADAYAPPAN, P., CHEN, Y., LIN, H., AND NGAI, T.-F. 2009. Data layout transformation for enhancing data locality on NUCA chip multiprocessors. In *Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques (PACT'09)*. IEEE Computer Society, Washington, DC, 348–357.

LUK, C.-K. AND MOWRY, T. C. 1999. Automatic compiler-inserted prefetching for pointer-based applications. *IEEE Transactions on Computers 48,* 2, 134–141.

MAGNUSSON, P. S., CHRISTENSSON, M., ESKILSON, J., FORSGREN, D., HALLBERG, G., HOGBERG, J., LARSSON, F., MOESTEDT, A., AND WERNER, B. 2002. Simics: A Full System Simulation Platform. *IEEE Computer 35,* 2.

OULD-AHMED-VALL, E.-M., DOSHI, K. A., YOUNT, C., AND WOODLEE, J. 2008. Characterization of SPEC CPU2006 and SPEC OMP2001: Regression Models and their Transferability. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'08)*. 179–190.

QURESHI, M., FRANCESCHINI, M., JAGMOHAN, A., AND LASTRAS, L. 2012. PreSET: Improving read-write performance of phase change memories by exploiting asymmetry in write times. In *Proceedings of the International Symposium on Computer Architecture (ISCA'12)*.

QURESHI, M. K., FRANCESCHINI, M. M., AND LASTRAS-MONTANO, L.-A. 2010. Improving read performance of phase change memories via write cancellation and write pausing. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'10)*.

RASQUINHA, M., CHOUDHARY, D., CHATTERJEE, S., MUKHOPADHYAY, S., AND YALAMANCHILI, S. 2010. An energy efficient cache design using spin torque transfer STT RAM. In *Proceedings of the 2010 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'10)*.

Scheuerlein, R., Gallagher, W., Parkin, S., Lee, A., Ray, S., Robertazzi, R., and Reohr, W. 2000. A 10 ns read and write non-volatile memory array using a magnetic tunnel junction and FET switch in each cell. In *Proceedings of the IEEE International Conference on Solid-State Circuits (ISSCC'00)*. 128–129.

Shivakumar, P. and Jouppi, N. P. 2001. *CACTI 3.0: An Integrated Cache Timing, Power, and Area Model*. Technical Report.

Singhee, A. and Rutenbar, R. A. 2009. Statistical blockade: Very fast statistical simulation and modeling of rare circuit events and its application to memory design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 28,* 8, 1176–1189.

Smullen, C. W., Mohan, V., Nigam, A., Gurumurthi, S., and Stan, M. R. 2011. Relaxing non-volatility for fast and energy-efficient STT-RAM caches. *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'11).*

Smullen IV, C. W., Mohan, V., Nigam, A., Gurumurthi, S., and Stan, M. R. 2011. Relaxing Non-Volatility for Fast and Energy-Efficient STT-RAM Caches. *Proceedings of the 2011 IEEE International Symposium on High Performance Computer Architecture (HPCA'11).*

Sun, G., Dong, X., Xie, Y., Li, J., and Chen, Y. 2009. A novel architecture of the 3D stacked MRAM L2 cache for CMPs. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'09)*. 239–249.

Sun, Z., Bi, X., Li, H., Wong, W.-F., Ong, Z.-L., Zhu, X., and Wu, W. 2011. Multi retention level STT-RAM cache designs with a dynamic refresh scheme. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'11)*.

Wen, W., Zhang, Y., Chen, Y., Wang, Y., and Xie, Y. 2012. PS3-RAM: A fast portable and scalable statistical STT-RAM reliability analysis method. In *Proceedings of the Design Automation Conference (DAC'12)*. 1187–1192.

Wolf, M. E. 1992. *Improving Locality and Parallelism in Nested Loops*. Ph.D. Dissertation. Stanford, CA. UMI Order No. GAX93-02340.

Wu, X., Li, J., Zhang, L., Speight, E., Rajamony, R., and Xie, Y. 2009. Hybrid cache architecture with disparate memory technologies. In *Proceedings of the International Symposium on Computer Architecture (ISCA'09)*.

Zhang, Y., Wang, X., Li, Y., Jones, A. K., and Chen, Y. 2012a. Asymmetry of MTJ switching and its implication to STT-RAM designs. In *Proceedings of the Design, Automation & Test in Europe (DATE'12)*. 1313–1318.

Zhang, Y., Wen, W., and Chen, Y. 2012b. The Prospect of STT-RAM Scaling from Readability Perspective. *IEEE Transactions on Magnetics 48,* 11, 3035–3038.

Zhou, P., Zhao, B., Yang, J., and Zhang, Y. 2009. Energy reduction for STT-RAM using early write termination. In *Proceedings of ICCAD*.

Received June 14, 2013; revised September 13, 2013; accepted November 13, 2013