A Novel Cache-Utilization-Based Dynamic Voltage-Frequency Scaling Mechanism for Reliability Enhancements

Yen-Hao Chen, Yi-Lun Tang, Yi-Yu Liu, Allen C.-H. Wu, and TingTing Hwang

***Abstract*— We propose a cache architecture using a 7T/14T SRAM (Fujiwara *et al.*, 2009) and a control mechanism for reliability enhancements. Our control mechanism differs from conventional dynamic voltage-frequency scaling (DVFS) methods in that it considers not only the cycles per instruction behaviors but also the cache utilization. To measure cache utilization, a novel metric is proposed. The experimental results show that our proposed method achieves 1000 times less bit-error occurrences compared with conventional DVFS methods under the ultralow-voltage operation. Moreover, the results indicate that our proposed method surprisingly not only incurs no performance and energy overheads but also achieves on average a 2.10% performance improvement and a 6.66% energy reduction compared with conventional DVFS methods.**

***Index Terms*— 7T/14T SRAM, cache, dynamic voltage-frequency scaling (DVFS), reliability.**

I. Introduction

RELIABILITY has become one of the major considerations in system design recently. Reliability is especially critical in systems where safety of human life is concerned. For example, an engine control unit (ECU) in an automotive system controls the fuel injection to the cylinders. A precise amount of fuel must be injected to the cylinders at an exact time, or the engine may run into dangerous conditions. In addition, an implantable pacemaking system [2] is expected to provide a perfectly stable function in order to maintain the health of the patient.

On the other hand, with the rapid progress of the advances in fabrication technology, more and more transistors can be embedded in a single chip. Hence, recent processor design developers tend to add more CPU cores as well as large caches into a single chip for performance improvements. This design trend has incurred a severe power-consumption problem that may greatly affect the battery life, performance, and reliability of the devices. In recent years, the dynamic voltage-frequency scaling (DVFS) technique has been used to reduce the power consumptions of processors [3]. Using this technique, a DVFS controller can detect computational patterns in execution and determine when to scale down the voltage and frequency of the CPU cores for energy savings. Many modern processor designs have supported per-core DVFS, e.g., ARM's big.LITTLE cores and Intel's Turbo Boost technology, in which each core has the ability to independently scale its voltage-frequency level. In recent multicore systems, extended control mechanisms are applied to control cores as well as uncore components for power reduction [4], [5]. Nevertheless, all of the above-mentioned DVFS frameworks suffer the limitation of the lowest supply voltage constraint due to the reliability of the SRAM cache, and none of them consider cache reliability in their control mechanisms.

A large cache in a processor generates excessive power consumption. Since the leakage power dominates the power consumption of a cache, scaling down the supply voltage seems to be a good solution for power reduction. However, many researchers also found that under the ultralow-voltage operation, the SRAM cache is very sensitive to noises and becomes very unreliable [6]. Over the years, many alternative SRAM cells have been proposed to tolerate low supply voltages, including 8T, 9T, 10T, and 12T SRAM cells [7]–[11], by trading off the SRAM cache density for better reliability. However, those alternative SRAM cells may entail performance loss when executing cache capacity-intensive applications. To address capacity and reliability issues, Fujiwara *et al.* [1] have proposed a configurable and dependable 7T/14T SRAM cache architecture that can trade cache capacity for reliability improvement under an ultralow supply voltage condition dynamically.

In this paper, we present a reliable cache architecture using the 7T/14T SRAMs and a control mechanism for DVFS by considering performance improvement, power reduction, and reliability enhancement. We study the interrelationship between computational patterns in execution and cache behaviors and devise a control scheme that can effectively switch the cache operation mode between a reliable state under an ultralow supply voltage for energy savings, a high-performance state under low cache utilization, and a normal state.

The rest of this paper is organized as follows. Section II describes previous work. Section III presents our motivation. In Sections IV and V, we present the proposed system

Manuscript received May 20, 2016; revised August 23, 2016; accepted September 23, 2016. Date of publication October 28, 2016; date of current version February 22, 2017.

Y.-H. Chen and T. Hwang are with the Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan (e-mail: yhchen@cs.nthu.edu.tw; s102062607@m102.nthu.edu.tw; tingting@cs.nthu.edu.tw).

Y.-L. Tang is with the Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan, and also with Novatek, Hsinchu.

Y.-Y. Liu is with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan 32003, Taiwan (e-mail: yyliu@saturn.yzu.edu.tw).

A. C.-H. Wu is with the School of Digital Media, Jiangnan University, Wuxi 214122, China (e-mail: allenwuuw@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2016.2614993

architecture and control mechanism, respectively. Finally, we present the experimental results and conclusions in Sections VI and VII.

II. Previous Work

Over the years, many alternative SRAM cells have been proposed to tolerate low supply voltages, including 8T SRAM cells [7], 9T SRAM cells [8], 10T SRAM cells [9], and 12T SRAM cells [10], [11], by trading off the SRAM cache density for better reliability. However, those alternative SRAM cells may suffer performance loss when executing cache capacity-intensive applications.

Another widely used technique for fault tolerance under low supply voltages is to use error-correction codes (ECCs). Nowadays, SRAM caches are typically equipped with single-error correction double-error detection (SECDED) codes [12], [13]. Chishti *et al.* [14] have proposed multibit segmented ECC, which divides a cache line into multiple small segments and corrects errors on a per-segment basis to simplify ECC implementation. Alameldeen *et al.* [15] used variable-strength ECC to address multibit failures of a small number cache lines. Zhang *et al.* [16] utilized aggressive double-error correction triple-error detection codes to achieve high reliability. However, using ECC requires additional hardware for encoding and decoding, which results in a larger cache area and longer access latency.

Data block replication for reliability enhancement has also been proposed. Chakraborty *et al.* [17] have proposed multicopy cache that maintains multiple copies of every data item to ensure reliability under low voltage. Yalcin *et al.* [18] presented a cache architecture that uses various replications to provide different degrees of fault tolerance. However, both of the above-mentioned replication approaches require massive area overhead.

Reconfiguration of caches to enhance reliability was also adopted by many researchers. Abella *et al.* [19] observed that most of the SRAM cells are robust enough to tolerate low supply voltages. Thus, they presented a fault tolerance scheme in caches by disabling unreliable SRAM cells. Ansari *et al.* [20] proposed the Archipelago cache architecture that groups several cache lines with the same wordlines and applies an MUXing layer to dynamically reconfigure itself to absorb failing SRAMs at block granularity. Mahmood *et al.* [21] also proposed a nonintrusive and reconfigurable cache named Macho that remaps faulty SRAM cells with word granularity. It also added an additional fault map in the cache tag to identify the faulty words. Sasan *et al.* [22] proposed to use auxiliary structures to improve defect tolerance. All of the above-mentioned cache architectures required an SRAM cell testing procedure on booting as well as a faulty table and complex remapping controller to record and tolerate defects, which will lengthen the cache access latency.

Instead of adopting a complex reconfiguration scheme, many studies proposed to disable faulty cache blocks and use only robust cache blocks to guarantee cache functionality [13], [16], [23]. However, those methods suffer inevitable performance penalty.

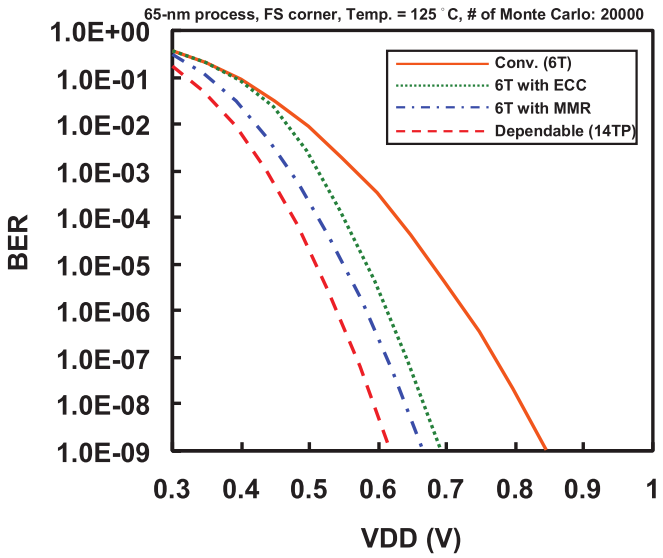


Fig. 1. Comparisons of BERs among the conventional methods and the dependable cache [1].

Many studies have been performed on a DVFS controlling framework. Most of them focus on single-core DVFS. Magklis *et al.* [24] proposed a method that uses a profile-based compiler and configuration instructions to scale up/down the voltage and frequency of the system. Isci *et al.* [25] proposed a hardware prediction table to predict memory loading and determine the voltage-frequency level of the application. Dhiman and Rosing [26] proposed a software-level DVFS framework that detects the memory usage of the cycles per instruction (CPI) to determine the voltage-frequency scaling level. DVFS on the interconnect network has also been studied (both links [4] and routers [5]). Chen *et al.* [27] propose to consider both the network and the last level cache (LLC) (so-called the uncore) when determining the network voltage-frequency level.

All of the above-mentioned DVFS frameworks suffer the limitation of the lowest supply voltage constraint of the reliable SRAM cache. Moreover, none of them consider the cache capacity requirements of workloads.

Works concerning real-time systems [28], [29] have studied the relationship between cache utilization and CPU utilization, I/O operations, memory access, or interrupts to satisfy hard timing constraints. However, reliability and cache power consumption were not considered.

III. Motivation

Fig. 1 shows the comparisons of bit-error rates (BERs) among the conventional methods and the dependable cache produced by Fujiwara *et al.* [1] where they have demonstrated that using the proposed 7T/14T memory cells, the minimum voltages in read and write operations are improved by 0.2–0.26 V, respectively. Furthermore, the BERs using the dependable low-power mode are much more reliable compared with the conventional 6T, 6T with ECC [30], and 6T with multimodule redundancy methods [31]. This design allows the memory cells to perform reliable operations under ultralow

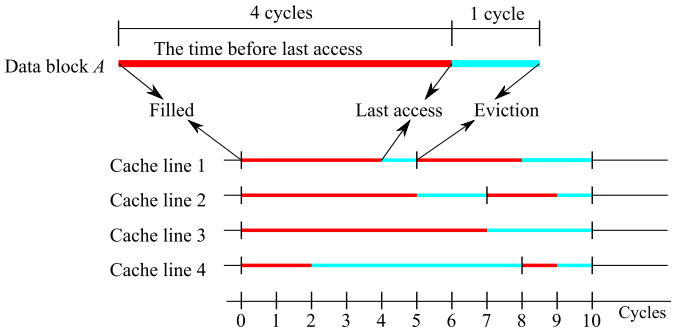


Fig. 2. Cache utilization example.

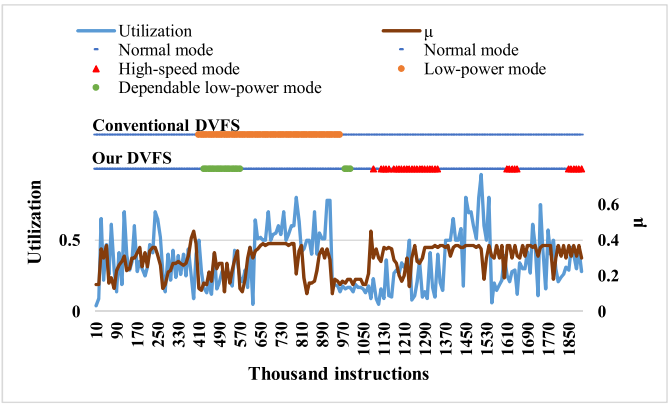


Fig. 3. Cache utilization and the conventional DVFS decisions of 435.gromacs.

supply voltages, and hence achieve higher energy savings. This motivates us to develop a cache system using the 7T/14T memory cells.

Consider the 7T/14T memory cells that consist of three modes: normal mode, high-speed mode, and dependable low-power mode (we will present the details of 7T/14T memory cells in Section IV-A). When operating in the high-speed mode and dependable low-power mode, the capacity of the memory cell is only half of that when operating in the normal mode. In this case, it is called the *line-merged cache*. In a *line-merged cache*, if a normal voltage is applied, we have a high-speed cache (high-speed mode) whereas if a low voltage is applied, we have a dependable low-power cache (dependable low-power mode). Consequently, if the cache utilization of the program and data is 50% or less, we can aggressively merge lines to achieve high speed or better reliability as well as energy savings. This motivates us to develop a control mechanism by investigating the cache utilization.

Let us first define cache utilization as follows:

$$Cache\ utilization \equiv \frac{1}{B \times T} \sum_{i=1}^{N} U_i \tag{1}$$

where U_i denotes the cycles between the ith data block being filled and the last access time of the data block, N is the number of data blocks evicted in time period T, and B is the total number of cache lines. For example, in Fig. 2, data block A is filled at time 0 and evicted at time 5. The last access time of cache block A is at time 4. Hence, the data block utilization of A is $4/5 = 80\%$. In addition, there are four cache lines and seven data blocks between the time period 0 and 10. The total useful time for each data block is the summation of the time between the filling and the last access time before eviction of each block, i.e., $4+3+5+2+7+2+1 = 24$. The total time of each block is the number of cache lines times the time period, $4 \times 10 = 40$. Thus, the overall cache utilization is $24/40 = 60\%$. If the data block utilization is high, the data block is useful during its lifetime. Otherwise, the data block does not improve the system performance but only sits idle and waits to be evicted by the cache controller.

Fig. 3 shows the relationships among cache utilization, μ, and operation modes of the 435.gromacs example. The two lines near the top in Fig. 3 are DVFS decisions made by two schemes, conventional DVFS [26] and ours. The conventional DVFS [26] determines whether the system is running in normal mode or low-power mode by a weighted value, while our DVFS framework does so by cache utilization. The weight is based on the μ values computed by $\mu = ((CPI_{computational})/(CPI_{average}))$ [26]. A low μ value indicates that it is more toward a memory-bounded operation. In this case, we will assign a higher weight to the low-power mode, and hence, it will guide the controller to switch the system into low-power mode. For instance, in Fig. 3, the system switches from normal mode into low-power mode at 400 to 990 thousand instructions. However, this control scheme for a conventional cache may raise a reliability issue when using an ultralow supply voltage due to increasing BERs of the cache cells.

Now let us consider using a 7T/14T cache. When the cache utilization is below 50%, we can switch the cache into the dependable low-power mode for energy savings without losing reliability by applying low voltage or into the high-speed mode for performance improvements by applying normal voltage. However, when the cache utilization is over 50% and running on the dependable low-power or high-speed modes, the cache misses will increase due to the 50% capacity reduction of the cache. This will result in performance loss and also higher energy consumption. When using the conventional DVFS control mechanisms, the decision of the operation mode solely depends on CPIs without considering the cache utilization. Obviously, it is insufficient to apply the conventional DVFS control mechanisms to a 7T/14T cache.

As shown in Fig. 3, during the period of 650 to 900 thousand instructions, the cache utilization is higher than 50%. If a 7T/14T cache is running in low-power mode, the cache capacity will be reduced by half, which will incur higher cache misses. Hence, our DVFS scheme switches the cache from low-power mode to normal mode, as shown in the line labeled $Our\ DVFS$. Now, consider that when the cache is running in normal mode and the cache utilization is lower than 50% (e.g., from 1550 to 1690 thousand instructions in Fig. 3), we can more aggressively switch the system into high-speed mode for performance improvements. For the example shown in Fig. 3, we can achieve 23.50% in cache miss reduction, 14.74% performance improvements, and 20.21% more energy savings compared with the conventional DVFS control method [26].

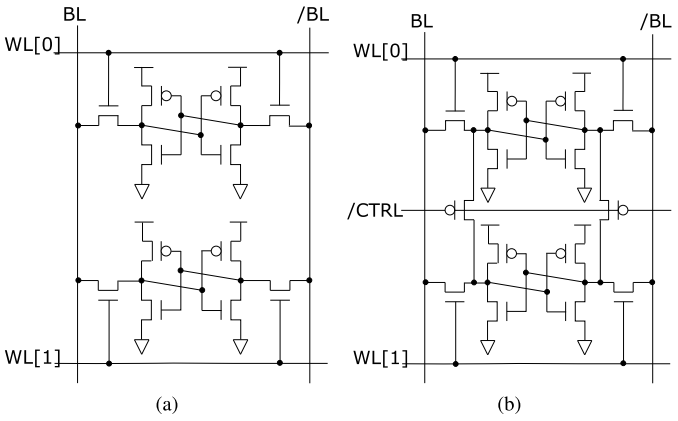


Fig. 4. Schematic of SRAM cells. (a) Two 6T SRAM cells. (b) 7T/14T SRAM cell.

From the above-mentioned observations, it motivates us to investigate the interrelationship between computational patterns in execution and cache utilization to devise a control scheme that can effectively switch the cache between normal, high-speed, and reliable low-power states under an ultralow supply voltage for energy savings without sacrificing performance.

IV. System Architecture

We first introduce the configurable 7T/14T dependable SRAM cache [1]. Then, we present the system modes and switching overheads.

A. 7T/14T SRAMs

Fig. 4(a) shows the schematic of two disjoint 6T SRAM cells. To access the data in the SRAM cell, one has to assert the corresponding wordline (WL[0] or WL[1]). Since the two SRAM cells share a single bitline (BL and /BL), the wordlines of these two SRAM cells (WL[0] and WL[1]) can only be asserted disjointly. Fig. 4(b) shows the schematic of a 7T/14T SRAM cell. Fujiwara *et al.* [1] have demonstrated that 7T/14T SRAM cells can be aligned in an array the same as the general 6T SRAM cell array. A 7T/14T SRAM cell consists of two conventional 6T SRAM cells and two additional pMOS transistors. By using the additional pMOS transistors, the 7T/14T SRAM can be operated at three different operation modes: normal mode, dependable low-power mode, and high-speed mode. In normal mode, a single 7T/14T SRAM cell is operated as two conventional 6T SRAM cells, and the L1 cache uses its full capacity. In the dependable low-power and high-speed modes, two 6T SRAM cells store only a single-bit value and the cache capacity is reduced to half. In dependable low-power mode, the 14T cell has a higher static-noise margin [6]. Hence, the overall SRAM system is much more reliable and able to tolerate lower system supply voltages. In this case, the CPU core operates at an ultralow supply voltage to achieve energy savings. In high-speed mode, the CPU operates at normal voltage, and high speed is achieved by driving a bitline with two transistors. In both the dependable low-power and high-speed modes, the cache sacrifices its capacity to obtain higher reliability or faster cache access latency. Table I summarizes the operations, capacity, and supply voltage of the 7T/14T SRAMs.

TABLE I
Operation Modes of 7T/14T SRAM Cache

feature \ mode	Normal	High-speed	Dependable low-power
Cache	regular	line-merged	
Capacity	full	half	half
Supply voltage	regular	regular	low

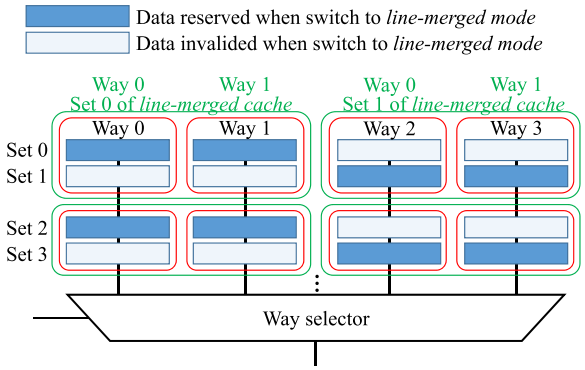


Fig. 5. Four-way way-variable 7T/14T SRAM cache.

Fig. 5 shows an example of a four-way way-variable 7T/14T SRAM cache. In general, each line in a cache set shares the same bitlines. Thus, two 6T SRAM cells of a 7T/14T SRAM cell will be used in adjacent sets. For instance, in Fig. 5, *set* 0 and *set* 1 share the same 7T/14T SRAM cells. In a regular cache, two 6T SRAM cells operate independently in two sets, but in a *line-merged cache*, they will be merged into one cell, as shown in the red rectangle, and the four-way cache becomes two-way. In Fig. 5, in order for the wordline to select sets, the system will designate *way* 0, *way* 1 to *set* 0 and *way* 2, *way* 3 to *set* 1 in *line-merged cache*, as shown in the green rectangle. One may be concerned that when switching modes from regular cache to *line-merged cache*, it will destroy both of the data blocks. Okumura *et al.* [32] have addressed this data block loss problem. They demonstrated that by carefully controlling the wordline and the two pMOS transistors (the $/CTRL$ signal in Fig. 4), they can achieve a reliable cache block copy simultaneously with a delay penalty of four cycles [32]. With the cache block copy technique, we can assume a 7T/14T SRAM cache scheme the same as in [33]. When switching from regular cache to *line-merged cache*, the 7T/14T SRAM cache copies *way* 0, *way* 1 data of *set* 0 (regular cache) to *set* 0 (*line-merged cache*), and *way* 2, *way* 3 data of *set* 1 (regular cache) to *set* 1 (*line-merged cache*). Note that when entering *line-merged cache*, half of the cache blocks will be destroyed (and need to writeback if they are dirty), while the other half of the cache blocks will be retained.

B. System Modes

Fig. 6 shows our proposed system architecture in which all CPU cores have a private L1 cache and share the single uniform LLC. The system is divided into five separated power

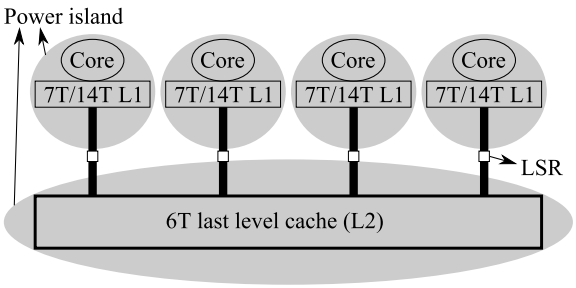


Fig. 6. Proposed system architecture.

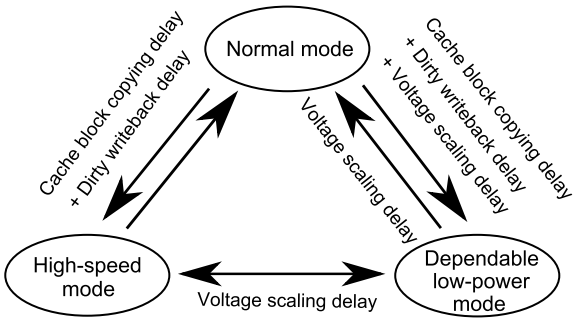


Fig. 7. System transition diagram.

islands where the voltage-frequency levels can be changed independently. The level shift registers (LSRs) are required to provide reliable signal transfers between different power islands with additional cycle delays. Because the system performance is sensitive to L1 cache latency, the CPU core and L1 cache often share the same voltage domain to avoid the LSR delay [5], [26], [27]. Since DVFS for the CPU core is usually aggressive, a more reliable SRAM for L1 cache is needed. Thus, we use the 7T/14T SRAM cells for L1 cache only and the traditional 6T SRAM for LLC.

In L1 cache, when low CPI and low cache utilization are detected, the controller will switch the 7T/14T L1 cache to the dependable low-power mode and evict half of its data blocks to ensure reliability. When only low utilization of the 7T/14T L1 cache is detected, the controller will switch the 7T/14T L1 cache to high-speed mode to trade its associativity for faster latency.

Fig. 7 shows the transition diagram of the system that consists of three switching delays: 1) dirty writeback delay caused by the eviction of half of the cache blocks; 2) voltage-frequency scaling delay; and 3) cache block copying delay caused by simultaneous block copies. When switching from the normal mode to the dependable low-power mode, it will incur three delays—cache block copying delay, dirty write-back delay, and voltage scaling delay—but only the voltage scaling delay will occur when switching on the other way around. In addition, only when switching from normal mode to high-speed mode will cache block copying delay and dirty writeback delay be incurred. Furthermore, when switching between dependable low-power mode and high-speed mode, both will incur voltage scaling delay. In our experiments, we have observed that most applications do not change phase frequently. Hence, the switching overhead is negligible and will not affect overall performance.

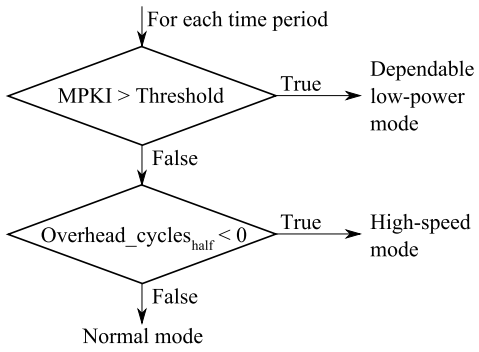


Fig. 8. Proposed cache utility-based voltage-frequency scaling mechanism.

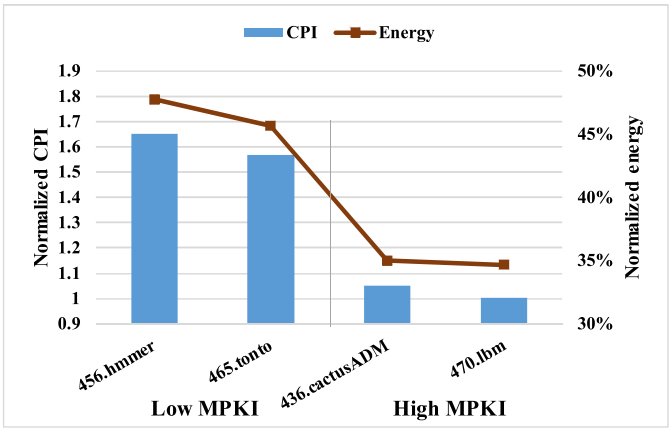


Fig. 9. Normalized CPI and energy of low/high MPKI applications.

V. Cache Utility-Based Voltage-Frequency Scaling Mechanism

In this section, we first present the proposed control mechanism in Section V-A. Then, in Sections V-B and V-C, we describe the hardware implementation of the control mechanism. Finally, the area and timing overhead analysis is given in Section V-D.

A. Proposed Control Mechanism

Our proposed control mechanism collects the information of the computational pattern in a predefined time period. Then, it uses the information to determine the 7T/14T cache operation mode of the next time period [34].

Fig. 8 shows the decision flowchart of the proposed control mechanism. We first use the collected information to determine whether it is a low CPI and low cache utilization computation. If it is, then the system enters the dependable low-power mode for energy savings. Otherwise, we determine whether it is only low cache utilization. If it is, the system enters the high-speed mode for performance improvements. This is explained in detail in the following.

In general, the cache blocks of high miss per kilo instructions (MPKI) applications do not have much data locality, which are accessed only once before the eviction, e.g., streaming. Thus, high MPKI applications usually have a low cache utilization (i.e., less than 50%). In addition, the performance of a high MPKI computational pattern is also not sensitive to the CPU core frequency. Fig. 9 shows the CPI and energy

comparisons of low and high MPKI applications with the system running in dependable low-power mode normalized to those in the normal mode. In Fig. 9, 456.hmmer and 465.tonto are low MPKI applications, and 436.cactusADM and 470.lbm are high MPKI applications. It shows that when running on the dependable low-power mode, the CPI of the low MPKI applications has increased by 65.05% and 56.74% while the CPI of the high MPKI applications has increased only 5.25% and 0.18%. Furthermore, the energy savings of the high MPKI applications are higher than that of the low MPKI applications due to the performance loss of the low MPKI applications. From the above-mentioned observations, we can switch the applications with a high MKPI to the dependable low-power mode for energy savings. We use a threshold to determine whether the applications are highly memory bounded, as shown in Fig. 8. We will discuss the threshold value in Section VI. Notice that high MPKI implies low CPI as well as low cache utilization. Hence, we use MPKI instead of CPI as the metric to determine whether to switch the system into the dependable low-power mode.

For some low MPKI applications with low cache utilization, higher driving capability of merged cells can be utilized to guarantee high performance. For this case, if the cache utilization is less than 50%, we sacrifice the cache associativity for performance by switching the system into the high-speed mode.

Now, let us consider how to obtain the cache utilization. Based on (1) in Section III, we need two counters to record the fill time and the last access time of each cache block. If we implement these two counters directly for each cache block, then $(2 \times the\ number\ of\ data\ blocks)$ counters are required. It will result in excessive hardware overheads. To alleviate this problem, we investigate another cache condition metric, $overhead\_cycles_{half}$, for cache utilization estimation. Equation (2) shows $overhead\_cycles_{half}$ defined as the cycle differences between the full-sized cache in the normal mode ($Cycles_{cost}$) and the half-sized cache in the high-speed mode ($Cycles_{benefit}$), as shown in (3) and (4). $Cycles_{cost}$ is estimated by the increased L1 miss count, $\Delta miss\_count_{L1}$, times the average L2 cache latency, avg_cycle_{L2}, while $Cycles_{benefit}$ is estimated by the read/write instruction count, IC_{rw}, times the reduced L1 cache latency, $\Delta cycle_{L1}$, as shown in (3) and (4)

$$Overhead\_cycles_{half} = Cycles_{cost} - Cycles_{benefit} \quad (2)$$

where

$$Cycle_{cost} = \Delta miss\_count_{L1} \times avg\_cycle_{L2} \quad (3)$$

and

$$Cycle_{benefit} = IC_{rw} \times \Delta cycle_{L1}. \quad (4)$$

The increased L1 miss count, $\Delta miss\_count_{L1}$, is the number of increased misses when the cache capacity is reduced to half. These misses will result in L2 accesses. Hence, $Cycle_{cost}$ is calculated as $\Delta miss\_count_{L1} \times avg\_cycle_{L2}$. Next, $\Delta cycle_{L1}$ is the number of cycles that can be saved when the cache capacity is reduced to half with a normal supply voltage. All read/write instructions to L1 cache will benefit from this cache cycle reduction. Hence, $cycle_{benefit}$ is calculated as $IC_{rw} \times \Delta cycle_{L1}$.

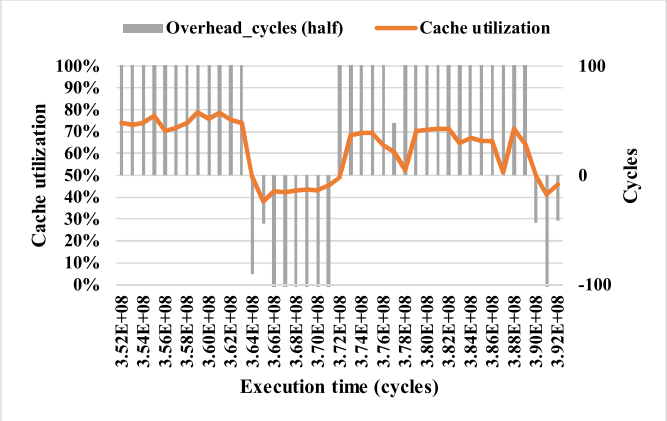


Fig. 10. Comparisons of the cache utilization of (1) and the $overhead\_cycles_{half}$ of (2) of 403.gcc.

We have studied the characteristics of two metrics of (1) and (2). Fig. 10 shows the relationships between the cache utilization metric by (1) and the $overhead\_cycles_{half}$ by (2). The results show that both metrics have the same indication. If cache utilization is higher than 50%, $overhead\_cycles_{half}$ is positive due to the additional misses on the L1 cache. Otherwise, if cache utilization is lower than 50%, $overhead\_cycles_{half}$ is negative. Consequently, we can use $overhead\_cycles_{half}$ to predict whether the cache utilization is higher or lower than 50%. This greatly simplifies our implementation that will be discussed in Section V-B.

B. Hardware Implementation

We can directly obtain the average L2 cache access latency, avg_cycle_{L2}, from the performance monitor counters that are built into the processors. The reduced L1 cache latency, $\Delta cycles_{L1}$, is given as a cache characteristic [26], while the read/write instruction count, IC_{rw}, is attainable online. However, to compute $\Delta miss\_count_{L1}$ an additional hardware, $\Delta miss\_counter$, is required.

In Section V-C, we discuss how to compute $\Delta miss\_count_{L1}$ in detail, consisting of two cases: regular cache (normal mode) and *line-merged cache* (dependable low-power or high-speed modes), where regular cache and *line-merged cache* denote that the L1 cache is in full and half capacity, respectively. The finite-state machine to implement an LRU replacement policy is modified to record $\Delta miss\_count_{L1}$. We use the following examples to explain how to compute $\Delta miss\_count_{L1}$.

Let the cache be four-way set associative. First, in a regular cache, an LRU sequence of four cache blocks is maintained by the LRU controller. Fig. 11 shows the LRU sequence transition examples where the LRU sequence from the most recently used block to the least recently used block is denoted as MRU, mru, lru, and LRU, respectively. Let us look at cases (1) and (2) shown in Fig. 11(a). If the memory request is a hit to MRU or mru blocks, this request will always be a hit regardless of the operation mode. In those cases, the LRU sequence is updated using the original LRU policy.

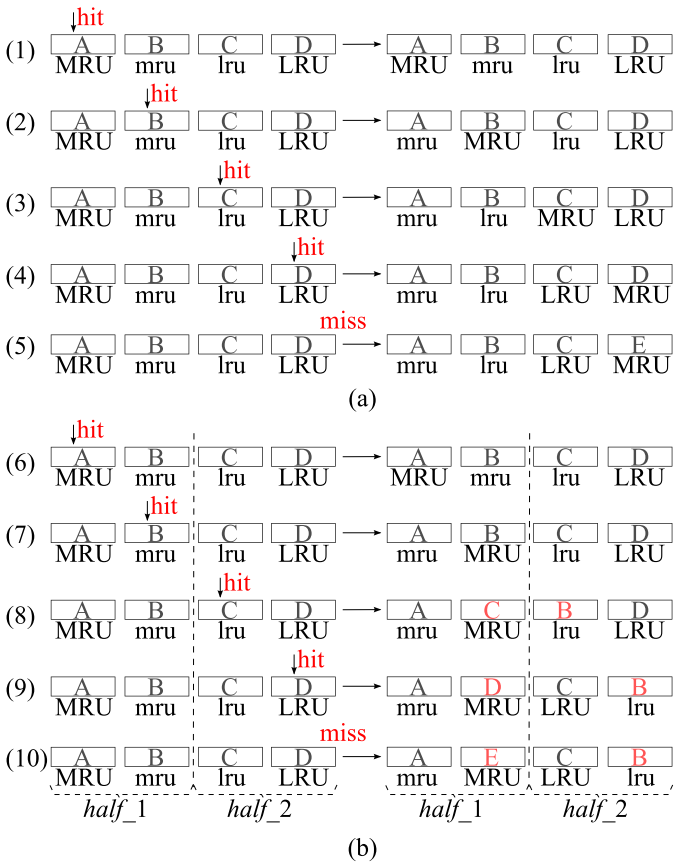


Fig. 11. Examples of LRU sequence and tag location transition. (a) Regular cache. (b) Line-merged cache.

Next, consider cases (3) and (4). If the memory request is a hit to the lru or LRU block, it means that this memory request is a hit in the regular cache but a miss in the *line-merged cache*. In those cases, besides updating the LRU sequence, $\Delta miss\_counter$ will be incremented by one to mimic this cache miss of *line-merged cache*. Finally, consider case (5). If the memory request is a miss, it means the data of this request are not in the cache. In this case, LRU block will be evicted and replaced by the requested block. In summary, in normal mode, the increased L1 miss count, $\Delta miss\_count_{L1}$, is obtained by counting the number of hits in the lru and LRU blocks.

Second, in a *line-merged cache*, the LRU controller only needs to maintain the LRU sequence for half of the tag array. Let us denote it as $half\_1$, as shown in Fig. 11(b). The other half of the tag array is not used and idle. Let us denote it as $half\_2$, as shown in Fig. 11(b). Since the tag array in $half\_2$ is not used, we can use this idle tag array to simulate the situation of the full-capacity tag array and record $\Delta miss\_count_{L1}$ occurred in the *line-merged cache*. The only difference is that the most recently used blocks (MRU and mru blocks) indicate the blocks in $half\_1$ and the least recently used blocks (lru and LRU blocks) indicate the blocks in $half\_2$ under the *line-merged cache*. Fig. 11(b) shows the different cases by using the idle tag array to simulate the full-capacity tag array in *line-merged cache*. Consider cases (6) and (7). If the memory request is a hit to the $half\_1$ blocks, it means this request will always be a hit regardless the operation mode. In those cases, the LRU sequence is updated using the original LRU policy. Next, let us look at cases (8) and (9). If the memory request is a hit to the lru block (the LRU block) in $half\_2$, it means the memory request is a hit in the regular cache but a miss in the *line-merged cache*. In those cases, $\Delta miss\_counter$ will be incremented by one to record this cache miss. The requested block will be fetched to the current location of mru block from the next level cache and LRU sequence is updated accordingly. In addition, to keep the LRU information of the full-capacity tag array, the tag data of the mru block in $half\_1$ must be copied into the tag location of the hit block, the lru block (the LRU block), in $half\_2$ by a temporal register, called tag_buffer (will be explained in Section V-C). Finally, consider case (10). If the request is a miss to L1, it means the memory request is always a miss in either regular cache or *line-merged cache*. In this case, the tag data of mru block in $half\_1$ will be copied into the tag_buffer, and then written to the tag location of current LRU block in $half\_2$. Then, the requested block will be fetched and replace the mru block in $half\_1$. Finally, the LRU sequence is updated accordingly. In summary, in *line-merged cache*, the increased L1 miss count, $\Delta miss\_count_{L1}$, is obtained by counting the number of hits in the $half\_2$.

From the above-mentioned discussion, it can be seen that due to the reduction of capacity, $\Delta miss\_count_{L1}$ can be obtained by utilizing the original tag array under the *line-merged cache*.

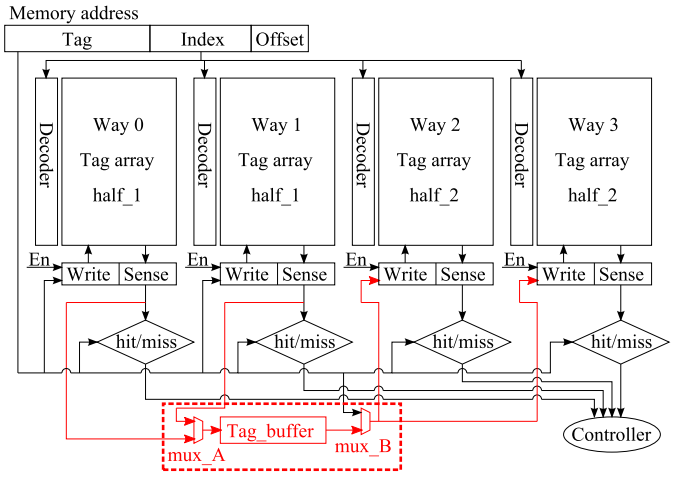


Fig. 12. Schematic of our proposed four-way tag array.

C. Tag Copying

In this section, we present the hardware modification necessary in order to compute the above $\Delta miss\_count_{L1}$. Fig. 12 shows our tag array and its control mechanism of a four-way tag array. Let us discuss how this architecture operates with a 7T/14T cache. There are three possible cases as follows.

First, the request is a hit or a miss in regular cache or the request is a hit to the $half\_1$ blocks in *line-merged cache*. In these cases, the tag array is updated by using the circuit built in hardware and an LRU policy. For instance, if it is

TABLE II
L1 CACHE CONFIGURATIONS

	6T L1 cache	7T/14T L1 cache			8T L1 cache	10T L1 cache	12T L1 cache
		Normal mode	Dependable low-power mode	High-speed mode			
CPU core voltage	0.8/0.75 V	0.8 V	0.55 V	0.8 V	0.8/0.55V	0.8/0.55V	0.8/0.55V
CPU core frequency	800/400 Mhz	800 Mhz	400 Mhz	800 Mhz	800/400 Mhz	800/400 Mhz	800/400 Mhz
L1 cache size	40KB	32KB	16KB	16KB	28KB	24KB	20KB
L1 cache associativity	10 way	8 way	4 way	4 way	7 way	6 way	5 way
L1 cache latency	4 cycles	4 cycles	4 cycles	3 cycles	4 cycles	4 cycles	4 cycles

a miss in normal mode, the controller will assert the write enable signal (the En signal in Fig. 12) of the LRU block and replace it with the new block. Finally, the controller updates the LRU sequence accordingly, as shown in cases (1)–(7) of Fig. 11.

Second, when the system operates in the *line-merged cache* and the request is a hit in the $half\_2$ blocks. In this case, before replacing the mru block (i.e., the least recently used block in $half\_1$) with the requested block, the tag data of the mru block should be copied to the tag location of the hit block in $half\_2$ in the following two steps. First, the controller stores the mru tag data in $half\_1$ into the tag_buffer by using the multiplexer, mux_A in Fig. 12. Then, the controller can write the tag data in tag_buffer to the designated tag location by mux_B. This write operation occurs at the same time as the tag data of the requested block are written to the tag location of the mru block. Finally, the LRU sequence is updated accordingly as shown in cases (8) and (9) in Fig. 11.

Third, when the system operates in the *line-merged cache* and the request is a miss in $half\_1$ and $half\_2$ block. In this case, before replacing the mru block in $half\_1$, the tag data are copied to the tag location of the current LRU block (i.e., the least recently used block in the $half\_2$ blocks) by the same tag copy.

D. Overhead Analysis

In our proposed architecture, we use the 7T/14T SRAM cache to implement the L1 data array. Compared with the traditional 6T SRAM, the area overhead of the 7T/14T SRAM cache is 16.67% more. On the other hand, compared with other reliable 8T SRAM designs that support ultralow-voltage operations [7], the area of the 7T/14T SRAM cache is 12.50% smaller.

The additional hardware to support our proposed control mechanism in the tag array includes a $\Delta miss\_counter$, a tag_buffer, and two multiplexers (mux_A and mux_B), which are relatively small and do not affect the overall die area. One may suspect that it may incur additional timing overheads due to the mru tag copying operation that transfers the tag data to tag_buffer. In fact, this step can be done in one tag access cycle delay. Furthermore, this tag copying is carried out when there is a miss and can be performed in parallel with the data fetching from the next level cache. As a result, there is no delay overhead for the implementation of our proposed control mechanism.

VI. ARCHITECTURE EVALUATIONS

In our experiment, we have evaluated the proposed control mechanism using the GEMS simulator [35] with the Linux kernel of version 2.6.15. We used the pipelined quad-core system with per-core DVFS. Table II shows the L1 cache configurations. Because the cell size of the 7T/14T SRAM is 16.67% larger than that of the conventional 6T SRAM, we have scaled the 7T/14T L1 cache size to 32KB to evaluate the performance and energy criteria under a fixed die area for cache. Similarly, for reliable caches composed of 8T, 10T, and 12T SRAM, we have scaled the L1 cache to 28, 24, and 20 kB, respectively. Moreover, a 2 MB, eight-way associative, 20 cycle latency LLC, and a 300 cycle latency DRAM were used in our environment. Other performance penalties of 7T/14T SRAM cache, including dirty writeback delay and cache block copying delay (described in Section IV-B), are all considered in the evaluation.

We have selected 17 applications from SPEC CPU2006 benchmarks [36] and randomly mixed them into 21 workloads. In addition, we added two workloads with four memory nonintensive applications (435.gromacs) and four memory intensive applications (470.lbm), which are mix10 and mix12, as shown in Table III. Due to limited space, we show only the results of the first 12 workloads in the figures. The rest of 11 workloads have similar energy/performance results. We set the last time period window to 1 million instructions, which is the same as in the previous work [26], [37]. It is because we have observed from the experiments that when executing the applications, the system does not frequently change the operation mode. It usually stays in an operation mode for a long period of tens of million instructions.

We have performed the evaluations of performance, power consumption, and reliability of our proposed system. The performance evaluation was estimated using the weighted speedup metric proposed by Kim *et al.* [38] as shown as follows:

$$Weighted\ speedup = \sum_i \frac{CPI_i^{alone}}{CPI_i^{shared}} \tag{5}$$

where CPI_i^{alone} is the CPI of core i when executing the single application alone on the platform and CPI_i^{shared} is the CPI of core i when sharing the platform resources with other applications. In addition, we have used the virtual platform power model proposed by Bartolini *et al.* [39] for the energy

TABLE III
WORKLOADS

Workloads	Applications
mix1	473.astar, 410.bwaves, 470.lbm, 437.leslie3d
mix2	436.cactusADM, 462.libquantum, 429.mcf, 470.lbm
mix3	473.astar, 450.soplex, 429.mcf, 470.lbm
mix4	403.gcc, 429.mcf, 450.soplex, 462.libquantum
mix5	433.milc, 471.omnetpp, 437.leslie3d, 453.povray
mix6	436.cactusADM, 456.hmmer, 410.bwaves, 462.libquantum
mix7	433.milc, 456.hmmer, 410.bwaves, 429.mcf
mix8	436.cactusADM, 456.hmmer, 410.bwaves, 437.leslie3d
mix9	403.gcc, 435.gromacs, 429.mcf, 437.leslie3d
mix10	435.gromacs, 435.gromacs, 435.gromacs, 435.gromacs
mix11	433.milc, 470.lbm, 429.mcf, 410.bwaves
mix12	470.lbm, 470.lbm, 470.lbm, 470.lbm
mix13	436.cactusADM, 471.omnetpp, 462.libquantum, 433.milc
mix14	453.povray, 483.xalancbmk, 437.leslie3d, 462.libquantum
mix15	462.libquantum, 470.lbm, 436.cactusADM, 453.povray
mix16	436.cactusADM, 462.libquantum, 470.lbm, 437.leslie3d
mix17	403.gcc, 435.gromacs, 473.astar, 462.libquantum
mix18	429.mcf, 450.soplex, 437.leslie3d, 462.libquantum
mix19	436.cactusADM, 471.omnetpp, 473.astar, 483.xalancbmk
mix20	436.cactusADM, 454.calculix, 462.libquantum, 465.tonto
mix21	453.povray, 454.calculix, 462.libquantum, 465.tonto
mix22	436.cactusADM, 462.libquantum, 473.astar, 410.bwaves
mix23	403.gcc, 462.libquantum, 435.gromacs, 456.hmmer

evaluation as shown as follows:

$$Core\ energy = (1 - idleness) * Energy_{active} + idleness * Energy_{idle} \quad (6)$$

$$Cache\ energy = usage_{access} * Energy_{access} + usage_{standby} * Energy_{standby}. \quad (7)$$

In (6) and (7), $idleness$ is used to model the deep sleep power state, $Energy_{active}$ and $Energy_{idle}$ are the energy dissipation in active and idle states, $Energy_{access}$ and $Energy_{standby}$ are the cache energy dissipation in access and standby states, and $usage_{access}$ and $usage_{standby}$ are the usage of cache in access and standby states. Furthermore, we used the L1 bit error count as the reliability metric in our reliability evaluation in which the error-rate model of 6T and 7T/14T SRAM cache was adopted from [1] with a 65-nm process and 125 °C temperature. A DVFS mechanism proposed by Dhiman and Rosing [26] was implemented. For comparisons, we have performed evaluations for the following configurations.

1) *Base:* The conventional configuration using 6T L1 cache without DVFS for reference.
2) *Online DVFS (0.55 and 0.75 V):* A DVFS mechanism using online learning [26] equipped with the given conventional 6T L1 cache configuration using high voltage (0.8 V) and low voltage (0.55 and 0.75 V).
3) *Reliable DVFS (8T, 10T, and 12T):* The online DVFS equipped with scaled 8T, 10T, and 12T reliable L1 cache configurations using high voltage (0.8 V) and low voltage (0.55 V).
4) *CUB-VFS:* The proposed cache utility-based voltage-frequency scaling (CUB-VFS) mechanism using the 7T/14T SRAM cache using high voltage (0.8 V) and low voltage (0.55 V).

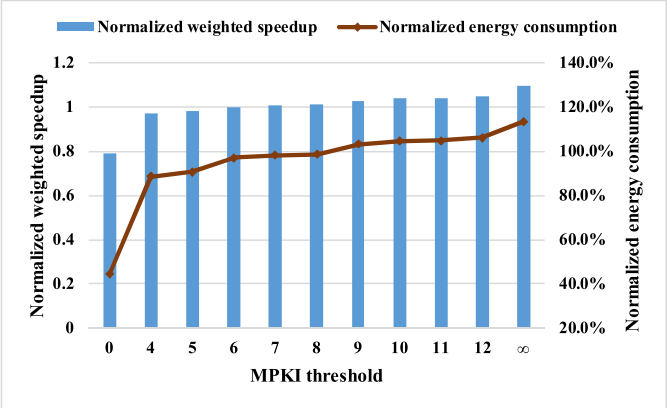


Fig. 13. Average performance/energy comparisons with various MPKI thresholds on all benchmarks.

A. Selection of MPKI Threshold in Control Mechanism

First, the MPKI threshold used in our control mechanism shown in Fig. 8 needs to be determined. We show the average normalized weighted speedup and energy consumption with various MPKI thresholds to that of the base configuration on all benchmarks in Fig. 13. Higher MPKI threshold means the cache stays in full capacity for a longer time. Intuitively, this will result in better performance with a penalty of more energy consumption. For the most energy-saving configuration (the leftmost bar in Fig. 13), our method can achieve 55.46% energy savings with 21.21% performance penalty. On the other hand, for the highest performance configuration (the rightmost bar in Fig. 13), our method achieves a 9.40% performance improvement with 13.57% more energy consumption. In order not to sacrifice performance, we select the MPKI threshold that results in the same performance as the base configuration. From Fig. 13, the performance of the MPKI threshold equal to 7 is the same as (or slightly better than) the base configuration. Thus, 7 is selected as the MPKI threshold for comparisons in the following experiments.

B. Reliability Evaluation

In this section, we compare the reliability of the conventional 6T cache configurations and ours operating at low-power (0.75 V) and ultralow-power (0.55 V) modes. Fig. 14 shows the total number of error blocks per day. The results show that, in low-power mode, the online DVFS using the safe supply voltage (0.75 V) can guarantee reliable operations (about six errors per day [40]). However, using the near-threshold voltage, the online DVFS (0.55 V) has very high error rates. On an average, it results in thousands of error blocks per day for all the workloads. On the other hand, our proposed method can achieve the error rate of about two error blocks per day and thus guarantee reliable operations using the near-threshold voltage.

Nowadays, modern processors typically are equipped with SECDED mechanisms [12], [13] to handle unavoidable soft errors. By using ECC redundant bits, systems can correct single-bit-error blocks and detect double-bit-error blocks. In other words, single-bit-error blocks can be safely corrected

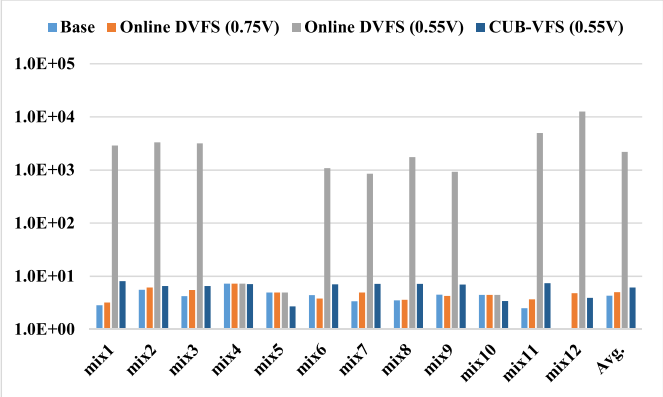


Fig. 14. Reliability comparisons by error blocks per day.

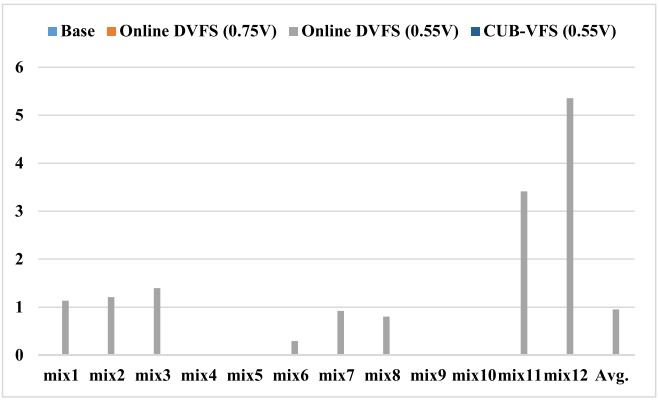


Fig. 15. Reliability comparisons by double-bit-errors per day.

without any performance overhead while double-bit-error blocks need roll back and reexecute instructions. Fig. 15 shows the comparisons of the number of double-bit-error blocks. Fig. 15 shows that only online DVFS (0.55 V) suffers the double-bit-error blocks.

Computing the expectation of multiple-bit-error blocks in ten years is also performed. The results are shown in Fig. 16. It shows that multiple-bit errors may occur in online DVFS (0.55 V). Multiple-bit-error blocks cannot be handled by the typically used SECDED mechanism. Although the probability of the multiple-bit-error blocks is very small, an error block may result in an incorrect execution of load instruction. If it does happen, the load instruction will further affect other instructions. To understand the effect of an incorrect load instruction, we have injected a load instruction with a multiple-bit-error block and recorded the next 100 000 instructions. Table IV shows the percentage of affected instructions. By Table IV, 12.97% of the instructions are affected after running 100 000 instructions. In a safety-critical system, such as ECU in automotive system and pacemaking system for health, this type of unreliable operation is not acceptable.

C. Compared With Conventional 6T SRAM Cache on Performance and Energy

In our *line-merged cache*, the cache capacity is reduced to half. In this section, we present the performance/energy evaluation to understand if the reliability enhancement of our mechanism sacrifices performance and energy.

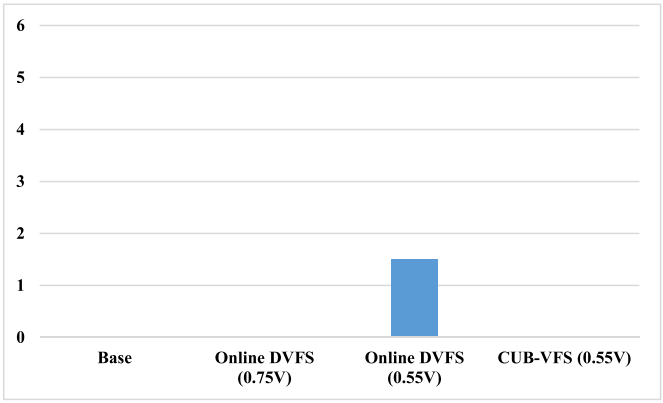


Fig. 16. Expectation of multiple-bit-error blocks in ten years.

TABLE IV
INSTRUCTIONS AFFECTED BY A SINGLE LOAD INSTRUCTION (462.libquantum)

Instruction type	Total	Unaffected	Affected
Load store inst.	22491	20433 (90.85%)	2058 (9.15%)
Arithmetic inst.	21965	19924 (90.71%)	2041 (9.29%)
Branch inst.	11117	11117 (100.00%)	0 (0.00%)
Stack inst.	217	216 (99.54%)	1 (0.46%)
Data manipulation inst.	44126	35254 (79.89%)	8872 (20.11%)
Others	84	84 (100.00%)	0 (0.00%)
Total	100000	87028 (87.03%)	12972 (12.97%)

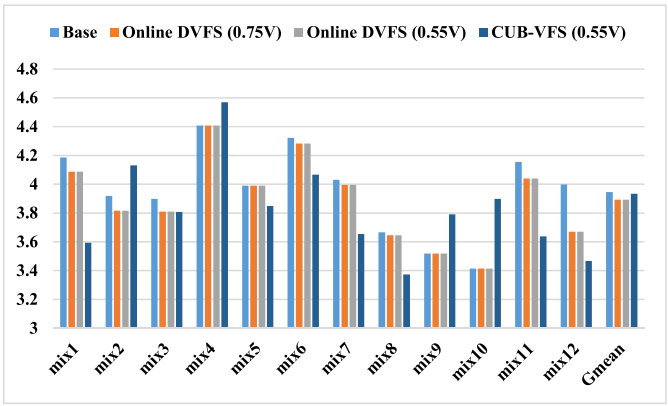


Fig. 17. Performance comparisons by weighted speedup.

Fig. 17 shows the performance comparison. On average, our proposed method achieves the same performance as the base configuration and outperforms the online DVFS configurations by 1.08%.

Fig. 18 shows the energy comparisons of all workloads normalized to the base configuration. The results show that our proposed method (CUB-VFS) achieves on average 5.51%, 2.97% more energy savings over the base configuration and the online DVFS using the safe supply voltage (0.75 V), and the same amount of energy savings with online DVFS using the near-threshold supply voltage (0.55 V).

Fig. 19 shows the breakdown of the total energy consumption. Because two 6T SRAM cells operate simultaneously

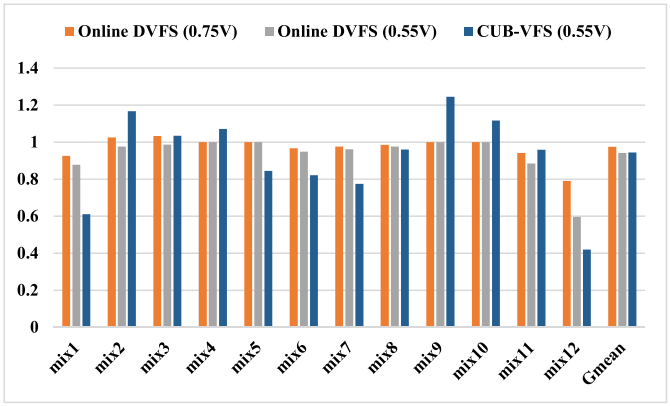


Fig. 18. Energy comparisons normalized to base.

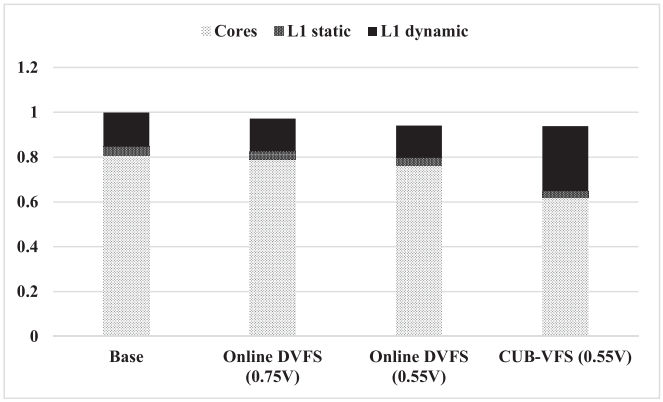


Fig. 19. Energy breakdown.

in *line-merged cache*, our proposed CUB-VFS consumes more dynamic energy than the others. However, our proposed method achieves more energy savings in the CPU core by more aggressively switching to the lower voltage-frequency mode. Thus, our overall system still achieves 5.51% energy savings as compared with the base configuration.

In our proposed architecture, entering *line-merged cache* may increase cache misses caused by dirty writeback and the reduction in capacity. Fig. 20 shows the detailed miss rate breakdown of our proposed mechanism. Overall, the increased miss rate caused by the capacity reduction and the writeback is 0.38% and less than 0.01%, respectively, the latter of which is too small to be shown in Fig. 20.

D. Compared With Other Reliable Caches

An 8T reliable SRAM cache has been proposed by Verma and Chandrakasan [7]. They used a sensing network with multiple sense amplifiers to achieve reliable operations under the near-threshold voltage. A 10T SRAM cache proposed by Chang *et al.* [9] utilizes a boosted wordline to ensure the worst case corner of write operations. Moore *et al.* [11] have proposed a 12T SRAM cache, which renounces cell density to ensure reliable operations under aggressive voltage scaling. Our reliable 7T/14T SRAM cell is designed by configuring two 7T cells into one 14T cell. In this section, we compare our method to other reliable caches in which low-power mode is operated at 0.55 V.

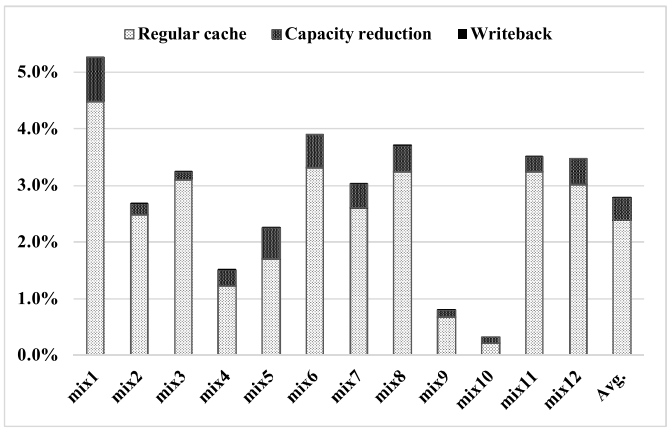


Fig. 20. Miss rate breakdown in *line-merged cache*.

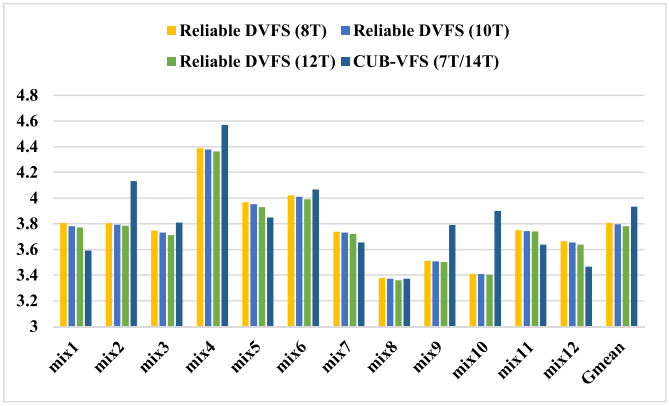


Fig. 21. Performance comparison between reliable caches.

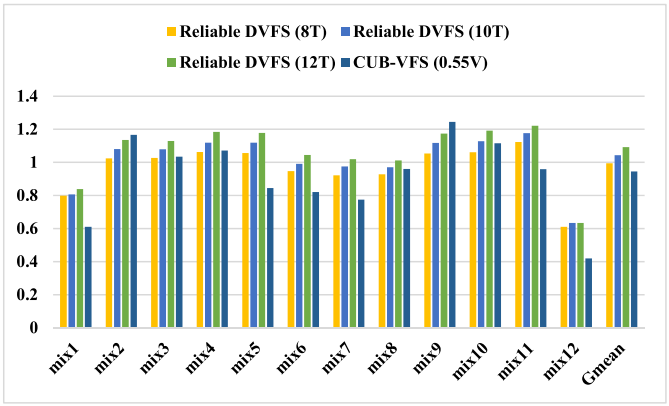


Fig. 22. Energy comparison between reliable caches.

Fig. 21 shows that using large reliable SRAM cells may result in a performance penalty because of capacity reduction. On the other hand, our proposed CUB-VFS method can reconfigure itself for different computational patterns. On an average, it achieves 3.32%, 3.66%, and 4.08% better performance improvements than those reliable 8T, 10T, and 12T SRAM caches, respectively.

Fig. 22 shows the energy comparison. All reliable SRAM caches suffer high dynamic energy consumption. In contrast,

our method is capable of dynamically adjusting itself under the computational pattern in execution. From the results, our proposed CUB-VFS achieves 4.88%, 9.82%, and 14.80% more energy savings compared with reliable 8T, 10T, and 12T SRAM caches, respectively.

VII. Conclusion

In this paper, we have presented a cache architecture and a control mechanism for reliability enhancements. Our proposed cache architecture uses 7T/14T SRAM [1] that can trade cache capacity for increased reliability as well as performance improvements under an ultralow supply voltage. We have investigated cache behaviors and devised a control mechanism by considering cache utilization. Our proposed control mechanism can greatly reduce bit-error occurrences and thus effectively control the cache to operate reliably without paying any speed or power-consumption penalties. We have performed a set of evaluations to demonstrate that our proposed method can outperform the conventional DVFS methods in reliability, power reduction, and performance.

References

[1] H. Fujiwara, S. Okumura, Y. Iguchi, H. Noguchi, H. Kawaguchi, and M. Yoshimoto, "A dependable SRAM with 7T/14T memory cells," *IEICE Trans. Electron.*, vol. E92-C, no. 4, pp. 423–432, 2009.

[2] S. Chede and K. Kulat, "Design overview of processor based implantable pacemaker," *J. Comput.*, vol. 3, no. 8, pp. 49–57, 2008.

[3] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," *IJCAET*, vol. 6, no. 4, pp. 440–459, 2014.

[4] L. Shang, L.-S. Peh, and N. K. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," in *Proc. 9th Int. Symp. High-Perform. Comput. Archit.*, 2003, pp. 91–102.

[5] A. K. Mishra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das, "A case for dynamic frequency tuning in on-chip networks," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, New York, NY, USA, Dec. 2009, pp. 292–303.

[6] E. Seevinck, F. J. List, and J. Lohstroh, "Static-noise margin analysis of MOS SRAM cells," *IEEE J. Solid-State Circuits*, vol. 22, no. 5, pp. 748–754, Oct. 1987.

[7] N. Verma and A. P. Chandrakasan, "A 256 kb 65 nm 8T subthreshold SRAM employing sense-amplifier redundancy," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 141–149, Jan. 2008.

[8] M.-H. Tu *et al.*, "A single-ended disturb-free 9T subthreshold SRAM with cross-point data-aware write word-line structure, negative bit-line, and adaptive read operation timing tracing," *IEEE J. Solid-State Circuits*, vol. 47, no. 6, pp. 1469–1482, Jun. 2012.

[9] I. J. Chang, J.-J. Kim, S. P. Park, and K. Roy, "A 32 kb 10T subthreshold SRAM array with bit-interleaving and differential read scheme in 90 nm CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC), Dig. Tech, Papers*, San Francisco, CA, USA, Feb. 2008, pp. 388–389.

[10] Y.-W. Chiu *et al.*, "40 nm bit-interleaving 12T subthreshold SRAM with data-aware write-assist," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 9, pp. 2578–2585, Sep. 2014.

[11] C. D. Moore, S. J. Keller, and A. J. Martin, "Ultra-low-power variation-tolerant radiation-hardened cache design," U.S. Patent 8 605 516, Dec. 10, 2013.

[12] K. Reick *et al.*, "Fault-tolerant design of the IBM Power6 microprocessor," *IEEE Micro*, vol. 28, no. 2, pp. 30–38, Mar. 2008.

[13] M. K. Qureshi and Z. Chishti, "Operating SECDED-based caches at ultra-low voltage with FLAIR," in *Proc. 43rd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Budapest, Hungary, Jun. 2013, pp. 1–11.

[14] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu, "Improving cache lifetime reliability at ultra-low voltages," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, New York, NY, USA, Dec. 2009, pp. 89–99.

[15] A. R. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, and S.-L. Lu, "Energy-efficient cache design using variable-strength error-correcting codes," in *Proc. 38th Int. Symp. Comput. Archit. (ISCA)*, San Jose, CA, USA, Jun. 2011, pp. 461–472.

[16] M. Zhang, V. M. Stojanovic, and P. Ampadu, "Reliable ultra-low-voltage cache design for many-core systems," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 59, no. 12, pp. 858–862, Dec. 2012.

[17] A. Chakraborty, H. Homayoun, A. Khajeh, N. Dutt, A. Eltawil, and F. Kurdahi, "E < MC^2: Less energy through multi-copy cache," in *Proc. Int. Conf. Compil., Archit., Synth. Embedded Syst. (CASES)*, Scottsdale, AZ, USA, Oct. 2010, pp. 237–246.

[18] G. Yalcin, A. Seyedi, O. S. Unsal, and A. Cristal, "Flexicache: Highly reliable and low power cache under supply voltage scaling," in *High Performance Computing* (Communications in Computer and Information Science), vol. 485, G. Hernández *et al.*, Eds. Valparaíso, Chile: Springer, Oct. 2014, pp. 173–190.

[19] J. Abella, J. Carretero, P. Chaparro, X. Vera, and A. González, "Low vccmin fault-tolerant cache with highly predictable performance," in *Proc. 42st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO-42)*, New York, NY, USA, Dec. 2009, pp. 111–121.

[20] A. Ansari, S. Feng, S. Gupta, and S. Mahlke, "Archipelago: A polymorphic cache design for enabling robust near-threshold operation," in *Proc. Int. Symp. High Perform. Comput. Archit.*, Feb. 2011, pp. 539–550.

[21] T. Mahmood, S. Kim, and S. Hong, "Macho: A failure model-oriented adaptive cache architecture to enable near-threshold voltage scaling," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2013, pp. 532–541.

[22] A. Sasan, H. Homayoun, A. M. Eltawil, and F. Kurdahi, "Inquisitive defect cache: A means of combating manufacturing induced process variation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 9, pp. 1597–1609, Sep. 2011.

[23] F. Hijaz, Q. Shi, and O. Khan, "A private level-1 cache architecture to exploit the latency and capacity tradeoffs in multicores operating at near-threshold voltages," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Asheville, NC, USA, Oct. 2013, pp. 85–92.

[24] G. Magklis, M. L. Scott, G. Semeraro, D. H. Albonesi, and S. Dropsho, "Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor," in *Proc. 30th Annu. Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA, 2003, pp. 14–27.

[25] C. Isci, G. Contreras, and M. Martonosi, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Washington, DC, USA, Dec. 2006, pp. 359–370.

[26] G. Dhiman and T. S. Rosing, "Dynamic voltage frequency scaling for multi-tasking systems using online learning," in *Proc. Int. Symp. Low Power Electron. Design*, Portland, OR, USA, Aug. 2007, pp. 207–212.

[27] X. Chen *et al.*, "In-network monitoring and control policy for DVFS of CMP networks-on-chip and last level caches," *ACM Trans. Design Autom. Electron. Syst.*, vol. 18, no. 4, Oct. 2013, Art. no. 47.

[28] C. Poellabauer, L. Singleton, and K. Schwan, "Feedback-based dynamic voltage and frequency scaling for memory-bound real-time applications," in *Proc. 11th IEEE Real Time Embedded Technol. Appl. Symp.*, Mar. 2005, pp. 234–243.

[29] X. Fu, K. Kabir, and X. Wang, "Cache-aware utilization control for energy efficiency in multi-core real-time systems," in *Proc. 23rd Euromicro Conf. Real-Time Syst. (ECRTS)*, Porto, Portugal, Jul. 2011, pp. 102–111.

[30] T. Suzuki, Y. Yamagami, I. Hatanaka, A. Shibayama, H. Akamatsu, and H. Yamauchi, "A sub-0.5-V operating embedded SRAM featuring a multi-bit-error-immune hidden-ECC scheme," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 152–160, Jan. 2006.

[31] C.-H. Chen and A. K. Somani, "Fault-containment in cache memories for TMR redundant processor systems," *IEEE Trans. Comput.*, vol. 48, no. 4, pp. 386–397, Apr. 2002.

[32] S. Okumura, S. Yoshimoto, K. Yamaguchi, Y. Nakata, H. Kawaguchi, and M. Yoshimoto, "7T SRAM enabling low-energy simultaneous block copy," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, San Jose, CA, USA, Sep. 2010, pp. 1–4.

[33] J. Jung, Y. Nakata, S. Okumura, H. Kawaguchi, and M. Yoshimoto, "Reconfiguring cache associativity: Adaptive cache design for wide-range reliable low-voltage operation using 7T/14T SRAM," *IEICE Trans. Electron.*, vol. E96-C, no. 4, pp. 528–537, 2013.

[34] H. Cook, M. Moretó, S. Bird, K. Dao, D. A. Patterson, and K. Asanovic, "A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness," in *Proc. 40th Annu. Int. Symp. Comput. Archit. (ISCA)*, Tel Aviv, Israel, Jun. 2013, pp. 308–319.

[35] M. M. K. Martin *et al.*, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *ACM SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, 2005.

[36] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, 2006.
[37] R. David, P. Bogdan, and R. Marculescu, "Dynamic power management for multicores: Case study using the Intel SCC," in *Proc. Int. Conf. VLSI Syst.-Chip*, 2012, pp. 147–152.
[38] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *Proc. 43rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Washington, DC, USA, Dec. 2010, pp. 65–76.
[39] A. Bartolini, M. Cacciari, A. Tilli, L. Benini, and M. Gries, "A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores," in *Proc. 20th Symp. Great Lakes Symp. VLSI (GLSVLSI)*, New York, NY, USA, 2010, pp. 311–316.
[40] Y. Nakata, S. Okumura, H. Kawaguchi, and M. Yoshimoto, "0.5-V operation variation-aware word-enhancing cache architecture using 7T/14T hybrid SRAM," in *Proc. 16th ACM/IEEE Int. Symp. Low Power Electron. Design (ISLPED)*, New York, NY, USA, Aug. 2010, pp. 219–224.

Yen-Hao Chen received the B.S. degree in computer science and engineering from Yuan Ze University, Taoyuan, Taiwan, in 2012, and the M.S. degree in computer science from National Tsing Hua University, Hsinshu, Taiwan, in 2014, where he is currently pursuing the Ph.D. degree with the Department of Computer Science.

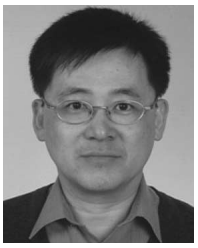
His current research interests include physical design automation and computer architecture.

Yi-Lun Tang received the B.S. and M.S. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2013 and 2015, respectively.

He is currently with Novatek, Hsinchu. His current research interests include computer architecture.

Yi-Yu Liu received the B.S. degree in physics, and the M.S. and the Ph.D. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 1997, 2000, and 2006, respectively.

He is currently an Associate Professor with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan, Taiwan. His current research interests include technology dependent logic synthesis, physical design automation, and computer architecture.



Allen C.-H. Wu received the B.S. degree in electronic engineering from the Taiwan Institute of Technology, Taipei, Taiwan, in 1983, the M.S. degree in electrical and computer engineering from The University of Arizona, Tucson, AZ, USA, in 1985, and the Ph.D. degree in computer science from the University of California at Irvine, Irvine, CA, USA, in 1992.

From 1992 to 2006, he was a Professor with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan. He is currently a Visiting Professor with the School of Digital Media, Jiangnan University, Wuxi, China.

TingTing Hwang received the M.S. and Ph.D. degrees in computer science from Pennsylvania State University, University Park, PA, USA, in 1986 and 1990, respectively.

She is currently a Professor with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan. Her current research interests include logic synthesis/optimization and high-level synthesis.