# A Novel Cache-Utilization Based Dynamic Voltage Frequency Scaling (DVFS) Mechanism for Reliability Enhancements

Yen-Hao Chen
Department of Computer Science
National Tsing Hua University, Taiwan
yhchen@cs.nthu.edu.tw

Yi-Lun Tang
Department of Computer Science
National Tsing Hua University, Taiwan
s102062607@m102.nthu.edu.tw

Yi-Yu Liu
Department of Computer Science and Engineering
Yuan Ze University, Taiwan
yyliu@saturn.yzu.edu.tw

Allen C.-H. Wu
School of Digital Media
Jiangnan University, China
allenwuuw@gmail.com

TingTing Hwang
Department of Computer Science
National Tsing Hua University, Taiwan
tingting@cs.nthu.edu.tw

## Abstract

*We propose a cache architecture using a 7T/14T SRAM [1] and a control mechanism for reliability enhancements. Our control mechanism differs from the conventional DVFS methods, which considers not only the CPI behaviors but also the cache utilizations. To measure cache utilization, a novel metric is proposed. The experimental results show that our proposed method achieves thousand times less bit-error occurrences compared to the conventional DVFS methods under the ultra-low voltage operation. Moreover, the results show that our proposed method surprisingly not only incurs no performance and energy overheads but also achieves on an average 5.1% performance improvement and 5% energy reduction compared to the conventional DVFS methods.*

## I. Introduction

In recent years, the dynamic voltage-frequency scaling (DVFS) technique has been proposed to reduce the power consumptions of processors [2]. Using this technique, a DVFS controller can detect the computational pattern in execution and determine when to scale-down the voltage and frequency of the CPU cores for energy saving. 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 up-and-down its voltage-frequency level. In recent multi-core systems, extended control mechanisms are applied to control cores as well as uncore components for power reduction [3]–[8]. Nevertheless, all of the above DVFS frameworks suffer from the limitation of the lowest supply voltage constraint due to the reliability of the SRAM cache and none of them takes into consideration of cache reliability in their control mechanisms.

A large cache in a processor generates excessive power consumption. Since the leakage power has dominated 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 ultra-low voltage operation, the SRAM cache is very sensitive to noises and becomes very unreliable [9]. Over the years, many alternative SRAM cells have been proposed to tolerate low supply voltages, including 8T, 9T, and 10T SRAM cells [10]–[13], by trading off the SRAM cache density for better reliability. However, those alternative SRAM cells may suffer from performance loss when executing cache capacity intensive applications. To address capacity and reliability issues, H. Fujiwara et al [1] have proposed a configurable 7T/14T dependable SRAM cache architecture that can tradeoff the cache capacity for the reliability improvement under an ultra-low 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 taking into consideration of performance improvement, power reduction, and reliability enhancement. We study the inter-relationship between the computational patterns in execution and the cache behaviors to devise a control scheme that can effectively switch the cache operation among a reliable state under an ultra-low supply voltage for energy saving, a high performance state under low cache utilizations, or a normal state.

## II. Motivation

Figure 1 shows the comparisons of BERs (Bit Error Rates) among the conventional methods and the dependable cache produced by H. Fujiwara et al [1]. They have demonstrated that using the proposed 7T/14T memory cells the minimum voltages in read and write operations are improved by 0.2V to 0.26V, respectively. Furthermore, the BERs using the dependable low-power mode are much reliable compared to the conventional 6T, 6T with ECC (Error Correction Code) [14], and 6T with MMR (Multi-Module Redundancy) methods [15]. This design allows the memory cells to perform reliable operations under ultra-low supply voltages, and hence achieve higher energy saving. This motivates us to develop a cache system using the 7T/14T memory cells.

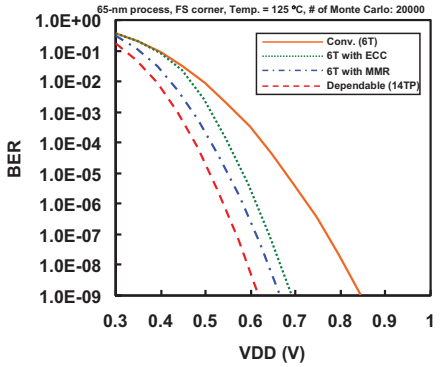


Fig. 1. The comparisons of BERs among the conventional methods and the dependable cache [1].

Consider the 7T/14T memory cells that consist of three modes: normal mode, high-speed mode and dependable low-power mode. When operating in the high-speed mode and dependable low-power mode, the capacity of the memory cell is only the 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 saving. This motivates us to develop a control mechanism by investigating the cache utilization.

978-3-9815370-7-9/DATE16/

Let us first define the cache utilization as below:

$$Cache\ utilization \equiv \frac{1}{B \times T} \sum_{i=1}^{N} U_i, \quad (1)$$

where $U_i$ denotes the cycles between the $i^{th}$ data block been filled and the last access time of the data block, $N$ the number of data blocks been evicted in the time period $T$, and $B$ the total number of cache lines. For example, in Figure 2, data block $A$ is filled at time 0 and evicted

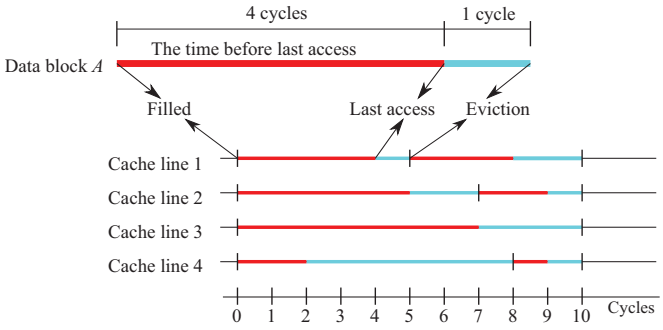


Fig. 2. A cache utilization example.

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 to 10. The total useful time for each data block is the summation of the time between filling and 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 life time. Otherwise, the data block does not improve the system performance but only sit idle and wait to be evicted by the cache controller.

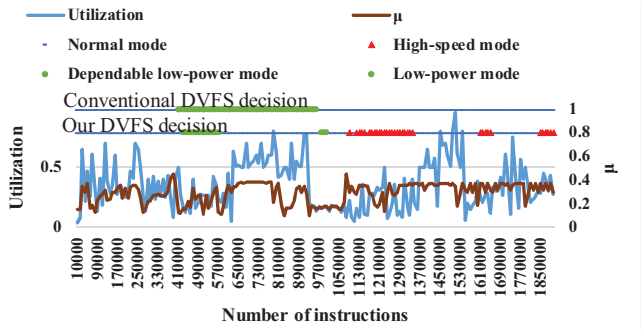


Fig. 3. The cache utilizations and the conventional DVFS decisions of 435.gromacs.

Figure 3 shows the relationships between cache utilization, $\mu$ value, and operation modes of the 435.gromacs example. We have applied the conventional DVFS [16] to determine whether the system is running on the normal mode or the low-power mode by assigning a weight value to each operation mode. The weight is based on the $\mu$ values computed by $\mu = \frac{CPI_{computational}}{CPI_{average}}$ [16]. A low $\mu$ value indicates that it is more toward a memory-bounded operation. In this case we will assign a higher weight value to the low-power mode and hence it will guide the controller to switch the system into the low-power mode. For instance, in Figure 3, the system switches from the normal mode into the low-power mode at instructions from 400000 to 990000. However, this control scheme for a conventional cache may raise a reliability issue when using an ultra-low supply voltage due to increasing BERs of the cache cells.

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

TABLE I
OPERATION MODES OF 7T/14T SRAM CACHE.

Now let us consider when using a 7T/14T cache and the cache utilization is below 50%, we can switch the cache into the dependable low-power mode for energy saving without losing the 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 more power consumption. When using the conventional DVFS control mechanisms, the decision of the operation mode solely depends on CPIs without taking into account the cache utilization. Obviously, it is insufficient to apply the conventional DVFS control mechanisms to a 7T/14T cache.

As shown in Figure 3, during the period of instructions 650000 to 900000, the cache utilization is higher than 50%. If a 7T/14T cache is running on the low-power mode, the cache capacity will be reduced by half that will incur higher cache misses. Hence, it will be better to switch the cache into the normal mode in this condition. Now, consider that when the cache is running on the normal mode and the cache utilization is lower than 50% (e.g. from instructions 1550000 to 1690000 in Figure 3), we can more aggressively switch the system into the high-speed mode for performance improvements. For the example shown in Figure 3, we can achieve 23.50% in cache-misses reduction, 14.74% performance improvements, and 20.21% power reduction compared to the conventional DVFS control method [16].

From the above observations, it motivates us to investigate the inter-relationship between the computational patterns in execution and the cache utilizations to devise a control scheme that can effectively switch the cache to normal, high-speed, and reliable low-power state under an ultra-low supply voltage for energy saving without sacrificing performance.

## III. System Architecture

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

A 7T/14T SRAM cell is adopted [1]. A 7T/14T SRAM cache can be operated at three different operation modes: normal mode, dependable low-power mode, and high-speed mode. In the normal mode, a single 7T/14T SRAM cell is operated as two conventional 6T SRAM cells. When operating in the normal mode, the L1 cache has the full capacity. In the dependable low-power mode, two 6T SRAM cells store only a single bit value, which has a higher static-noise margin (SNM) [9]. Hence, the overall SRAM system is much reliable and able to tolerate lower system supply voltages. In the dependable low-power mode, CPU core operates at an ultra-low supply voltage to achieve energy saving. In addition, when CPU operates at a normal voltage, the high-speed mode can achieve faster operation speed by driving a bitline with two transistors. In both the dependable low-power and high-speed modes, the cache tradeoffs 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.

Figure 4 shows our proposed system architecture in which we use the 7T/14T SRAM cells for the L1 cache and the 6T SRAM for the LLC (Last Level Cache). The system is divided into five separated power islands in which the voltage-frequency level can be changed independently. The LSRs (Level Shift Registers) are required to provide reliable signal transfers between different power islands with additional cycle delays. All CPU cores have a private L1 cache and share the single uniform LLC. When low CPI and low cache utilization are detected, the controller will switch the 7T/14T L1 cache to enter the dependable low-power mode and

evict half of its data blocks for ensuring the reliability. When only low utilization of the 7T/14T L1 cache is detected, the controller will switch the 7T/14T L1 cache to the high-speed mode to tradeoff its associativity for the faster latency.

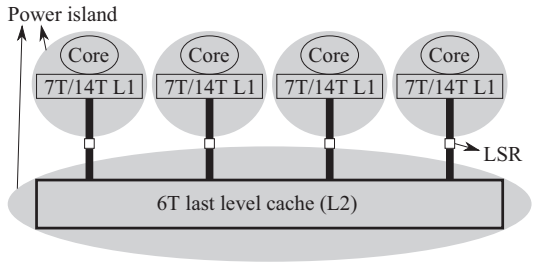


Fig. 4. The proposed system architecture.

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

## IV. Cache Utility-based Voltage-frequency Scaling Mechanism

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

### A. *The proposed control mechanism*

Figure 5 shows the proposed control mechanism. We first determine whether it is a low CPI and low cache utilization computation. If yes, then system enters the dependable low-power mode for energy saving. Otherwise, we determine whether it is only low cache utilization. If yes, the system enters the high-speed mode for performance improvements. We will explain it in details as below.

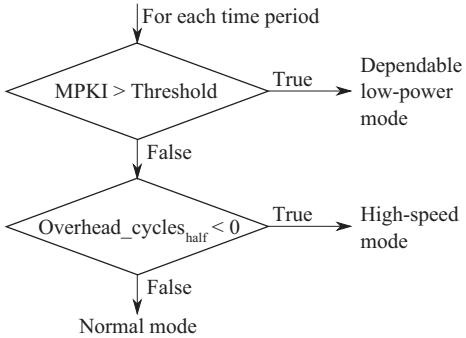


Fig. 5. The proposed cache utility-based voltage-frequency scaling mechanism.

In general, the cache blocks of high MPKI (Miss Per Kilo Instructions) applications do not have much data locality, which are accessed only once before the eviction, i.e. streaming. Thus, the 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. Figure 6 shows the normalized CPI (Cycles Per Instruction) and energy comparisons of low and high MPKI applications with system running on the normal mode and dependable low-power mode. It shows that when running on the dependable low-power mode, the performance of the low MPKI applications has degraded by 35%-40% while the performance of the high MPKI applications has degraded only less than 5%. Furthermore, the energy saving of the high MPKI applications is higher than that of the low MPKI applications due to the performance loss of the low MPKI applications. From the above observations, we can switch the applications with a high MKPI to the dependable low-power mode for energy saving. We use a threshold to determine whether the applications are highly memory bounded as shown in Figure 5. We will discuss the threshold value in Section V. 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.

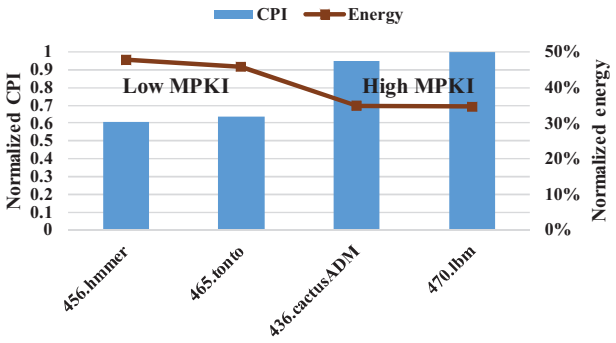


Fig. 6. The normalized CPI and energy of low/high MPKI applications.

For some low MPKI applications with low cache utilizations, those require high supply voltage-frequency level to guarantee the high performance. For this case, if the cache utilization is less than 50%, we can tradeoff the cache associativity to improve the performance by switching the system into the high-speed mode.

Now, let us consider how to obtain the cache utilization. Based on Equation (1) in Section II, 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 excessive hardware overheads problem, we investigate another cache condition metric, $overhead\_cycles_{half}$, for cache utilization estimation. Equation (2) shows the $overhead\_cycles_{half}$ that is 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 Equations (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 Equations (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 miss count when the cache capacity is reduced to half. These misses will result in the access to L2. Hence, $Cycle_{cost}$ is calculated as $\Delta miss\_count_{L1} \times avg\_cycle_{L2}$. Next, $\Delta cycle_{L1}$ is the number of cycles 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 $IC_{rw} \times \Delta cycle_{L1}$.

We have studied the characteristics of two metrics of Equation (1) and Equation (2). Figure 7 shows the relationships between the cache utilization metric by Equation (1) and the $overhead\_cycles_{half}$ by Equation (2). The results show that both metrics have the same indication. If the cache utilization is higher than 50%, the $overhead\_cycles_{half}$ is positive due to the additional misses on the L1 cache. Otherwise, if the cache utilization is lower than 50%, the $overhead\_cycles_{half}$ is negative. Consequently, we can use the $overhead\_cycles_{half}$ to predict whether the cache utilization is higher or lower than 50%. This can greatly simplify our implementation that will be discussed in Section IV-B.

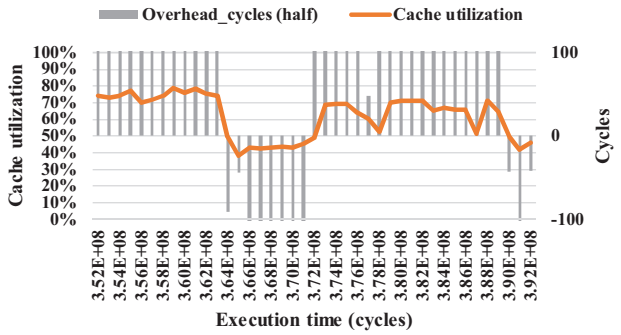


Fig. 7. The comparisons of the cache utilization of Equation (1) and the $overhead\_cycles_{half}$ of Equation (2) of 403.gcc.

### B. Hardware implementation

We can directly obtain the average L2 cache access latency, $avg\_cycle_{L2}$, from the PMCs (Performance Monitor Counters) which is built in the processors. The reduced L1 cache latency, $\Delta cycles_{L1}$, is given as a cache characteristic [16], while the read/write instruction count, $IC_{rw}$, is online attainable. However, to compute $\Delta miss\_count_{L1}$ an additional hardware, $\Delta miss\_counter$, is required.

In the following, we discuss how to compute $\Delta miss\_count_{L1}$ in details, 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 L1 cache is in full and half capacity, respectively. The finite-state machine to implement LRU replacement policy is modified to record $\Delta miss\_count_{L1}$. We use the following examples to explain how to compute the $\Delta miss\_count_{L1}$.

Let the cache be 4-way set associative. Firstly, in a regular cache, LRU sequence of 4 cache blocks is maintained by the LRU controller. Figures 8(a)-(e) show the LRU sequence transition examples where the LRU sequence from the most recently used block to the least recently used block are denoted as $MRU$, $mru$, $lru$, and $LRU$, respectively. Let us look at the cases shown in Figures 8(a)-(b). If the memory request is a hit to $MRU$ or $mru$ blocks, this request will always be a hit regardless the operation mode. In this case, the LRU sequence is updated using the original LRU policy. Next, consider the cases shown in Figures 8(c)-(d). If the memory request is a hit to the $lru$ or $LRU$ block, it means this memory request is a hit in the regular cache but a miss in the *line-merged cache*. In this case, besides updating the LRU sequence, $\Delta miss\_counter$ will be incremented by one to record this cache miss of *line-merged cache*. Finally, consider the case shown in Figure 8(e). If the memory request is a miss, it means the data of this request is not in the cache. In this case, $LRU$ block will be evicted and replaced by the requested block. Thus, 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.

Secondly, in a *line-merged cache*, 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 Figures 8(f)-(j). The other half of the tag array is not used and idle. Let us denote it as $half\_2$ as shown in Figures 8(f)-(j). 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*. Figures 8(f)-(j) show the different cases by using the idle tag array to simulate the full capacity tag array in *line-merged cache*. Consider the cases shown in Figures 8(f)-(g). 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 this case, the LRU sequence is updated using the original LRU policy. Next, let us look at the cases shown in Figures 8(h)-(i), 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 this case, the $\Delta miss\_counter$ will be incremented by one to record this cache miss. 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 in $half\_2$ by a temporal register, called $tag\_buffer$ (will be explained in Section IV-C). Also, the requested block will be fetched to the $mru$ block from the next level cache and LRU sequence is updated accordingly. Finally, consider the case shown in Figure 8(j). 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 $LRU$ block in $half\_2$. Then, the requested block will be fetched and replaced the $mru$ block in $half\_1$. Finally, the LRU sequence is updated accordingly. Thus, 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 discussion, it can be seen that $\Delta miss\_count_{L1}$ due to the reduction of capacity can be obtained by utilizing the the original tag array under the *line-merged cache*.

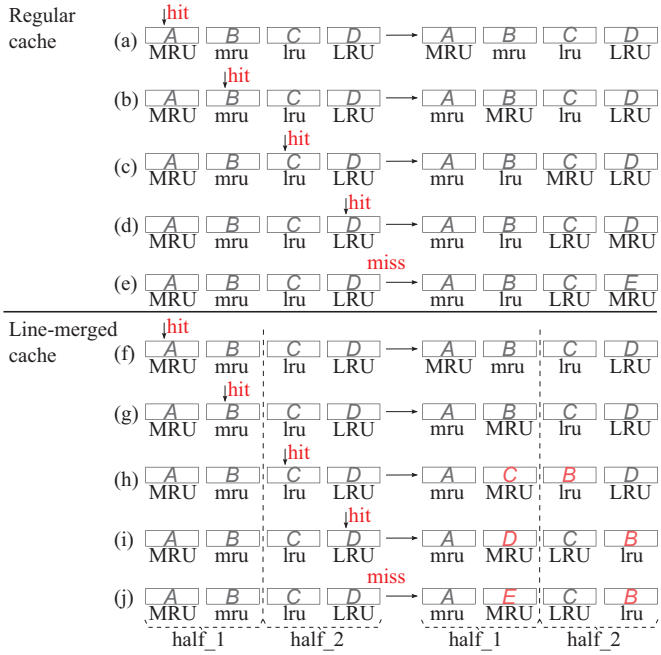


Fig. 8. Examples of LRU sequence and tag location transition.

### C. Tag copying

In this section, we present the hardware modification in order to compute the above $\Delta miss\_count_{L1}$. Figure 9 shows our tag array and

its control mechanism of a 4-way tag array. Let us discuss how this architecture operates with a 7T/14T cache. There are three possible cases as below:

Firstly, 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 LRU policy. For instance, if it is a miss in normal mode, the controller will assert the write enable signal (the $En$ signal in Figure 9) of the $LRU$ block and replace it by the new block. Lastly, the controller updates the LRU sequence accordingly as shown in Figures 8(a)-(g).

Secondly, 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 $half\_1$) by 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 Figure 9. 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 requested block is written to the tag location of the $mru$ block. Lastly, the LRU sequence is updated accordingly as shown in Figures 8(h)-(i).

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

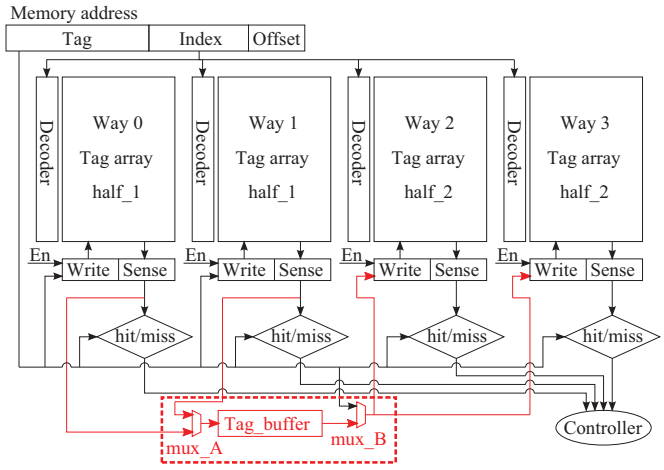


Fig. 9. The schematic diagram of our proposed 4-way tag array.

### D. Overhead analysis

In our proposed architecture, we use the 7T/14T SRAM cache to implement the L1 data array. Compared to the traditional 6T SRAM, the area overhead of the 7T/14T SRAM cache is 14.29%. On the other hand, compared to the other reliable 8T SRAM designs that support ultra-low voltage operations [10], [11], the area of the 7T/14T SRAM cache is 14.29% smaller.

The additional hardware to support our proposed control mechanism in the tag array includes only a $\Delta miss\_counter$, a $tag\_buffer$, and two multiplexers ($mux\_A$ and $mux\_B$), which is relatively small and does not affect the overall die area. One may suspect that, it may incur additional timing overheads due to the $mru$ tag copying operation that requires to transfer the tag data to $tag\_buffer$. 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.

| | Normal mode | Dependable low-power mode | High-speed mode |
|---|---|---|---|
| Supply voltage | 0.8 V | 0.55 V | 0.8 V |
| CPU core freq. | 800 Mhz | 400 Mhz | 800 Mhz |
| L1 cache size | 32KB | 16KB | 16KB |
| L1 cache assoc. | 8 way | 4 way | 4 way |
| L1 cache lat. | 4 cycles | 4 cycles | 3 cycles |
| L2 cache | 256KB, 8 way, 20 cycles | | |
| DRAM cycles | 300 | 150 | 300 |

TABLE II
ARCHITECTURE CONFIGURATIONS.

## V. EXPERIMENTAL RESULTS

In the experiments, we have evaluated the proposed control mechanism using the GEMS simulator [17] with the Linux kernel of version 2.6.15. We used the quad-core system with per core DVFS. Table II shows the L1 cache configurations. Because the cell size of the 7T/14T SRAM is 14.29% larger than the 6T SRAM, we have scaled the L1 cache size to 40KB for a fair comparison. In addition, we have selected 17 applications from SPEC CPU2006 benchmarks [18] and randomly mixed them into 20 workloads. In addition, we added two workloads with four memory non-intensive applications (435.gromacs) and four memory intensive applications (470.lbm) which are mix9 and mix11. From the experiments, we have observed that when executing the applications, the system did not frequently change the operation mode. They usually stay in an operation mode for a long period of tens of million instructions. From the above observations, we set the time period window to 1 million instructions, which is the same as in [5], [16].

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 Y. Kim et al. [19] as shown below: $Weighted\ speedup = \sum_i \frac{CPI_i^{alone}}{CPI_i^{shared}}$. In addition, we have used the power model proposed by A. Bartolini et al. [20] for the energy saving evaluation. Furthermore, we have used the L1 bit error count as the reliable metric in our reliability evaluation in which the error-rate model of 6T and 7T/14T SRAM cache was adopted from [1] with a 65nm process and 125° C temperature. We have also implemented a DVFS mechanism proposed by G. Dhiman and T. S. Rosing [16]. For comparisons, we have performed the following four experiments:

- **Reference :** The conventional configuration for reference.
- **Online DVFS:** A dynamic voltage-frequency scaling mechanism using the online learning [16] equipped with the given scaled 6T L1 cache configuration.
- **Online DVFS (0.55V):** the online DVFS equipped with the scaled 6T L1 cache configuration that can be scaled to the near-threshold voltage (0.55V).
- **CUB VFS:** The proposed cache utility-based voltage-frequency scaling mechanism using the 7T/14T SRAM cache which can be scaled down to 0.55V. The MPKI threshold is determined by the average of the one-third highest MPKI applications among the selected benchmarks.

Figure 10 shows the bit errors comparisons. The results show that the online DVFS using the safe supply voltage can guarantee reliable operations (about six errors per day [21]). However, using the near-threshold voltage, the online DVFS has very high error rates, on an average almost ten thousands errors per day for all the workloads. On the other hand, our proposed method can achieve the error rate of about two errors per day and thus guarantee reliable operations using the near-threshold voltage.

In our *line-merged cache*, the cache capacity is reduced to half. Next, we want to examine if there is any energy or performance overhead of our proposed method. Figure 11 shows the comparisons on the normalized energy comparisons. Unexpectedly, the results show that our proposed method (CUB VFS) achieves on an average 5% more energy saving over the online DVFS using the safe supply voltage (0.75V) and achieves the same energy saving compared to the online DVFS using the near-threshold supply voltage (0.55V). Finally, Figure 12 shows

the performance comparisons. Similarly, it turns out that our proposed method not only incurs no performance overhead but outperforms the conventional configuration and the online DVFS method by 2.6% and 5.1%, respectively. The results have shown that our new metric of cache utilization is indeed feasible and effective.

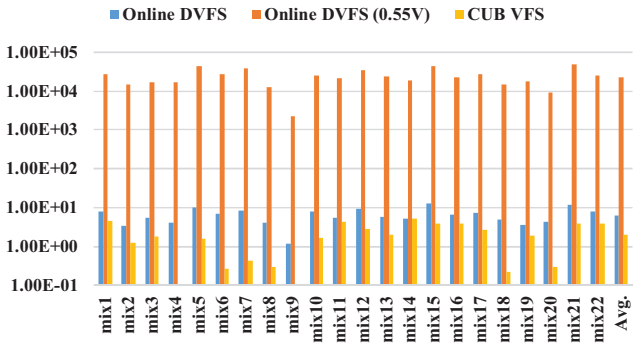


Fig. 10. The reliability comparisons by bit errors per day.

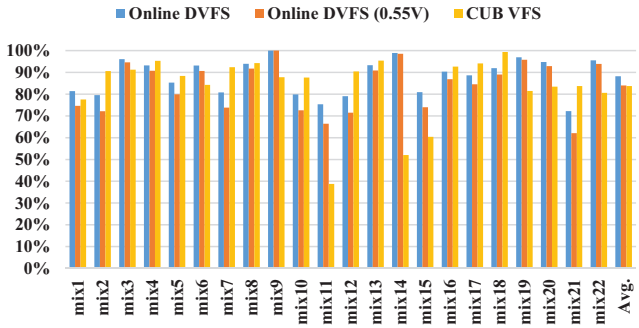


Fig. 11. The energy comparisons normalized to reference.

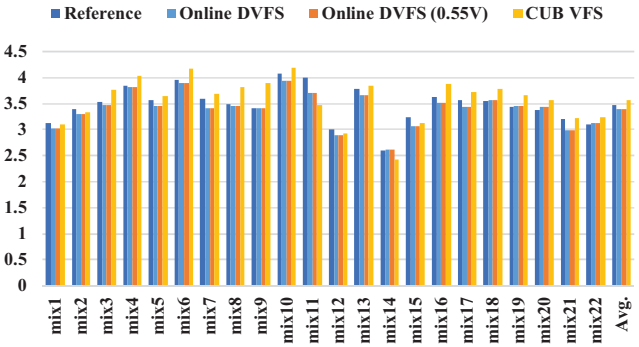


Fig. 12. The performance comparisons by weighted speedup.

## VI. Conclusions

In this paper, we present a cache architecture and a control mechanism for reliability enhancements. Our proposed cache architecture uses a 7T/14T SRAM [1] that can tradeoff the cache capacity for the reliability as well as performance improvements under an ultra-low supply voltage. We have investigated the cache behaviors and devised a control mechanism by taking into account the cache utilizations. Our proposed control mechanism can greatly reduce the bit-error occurrences and thus effectively control the cache to operate in a reliable state without paying any speed and power-consumption penalties. We have performed a set of experiments 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," in *ieice transactions on electronics*, pp. 423–432, 2009.

[2] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," *CoRR*, vol. abs/1401.0765, 2014.

[3] A. Bhattacharjee and M. Martonosi, "Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors," *SIGARCH Comput. Archit. News*, vol. 37, pp. 290–301, June 2009.

[4] N. Ioannou, M. Kauschke, M. Gries, and M. Cintra, "Phase-based application-driven hierarchical power management on the single-chip cloud computer," in *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques*, PACT '11, (Washington, DC, USA), pp. 131–142, IEEE Computer Society, 2011.

[5] R. David, P. Bogdan, and R. Marculescu, "Dynamic power management for multicores: Case study using the intel scc," in *International Conference on VLSI and System-on-Chip*, pp. 147–152, 2012.

[6] A. Bartolini, C. Hankendi, A. K. Coskun, and L. Benini, "Message passing-aware power management on man-core systems," *Journal of Low Power Electronics*, pp. 1–19, 2014.

[7] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, "Bounding energy consumption in large-scale mpi programs," in *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, SC '07, (New York, NY, USA), pp. 49:1–49:9, ACM, 2007.

[8] P. Bogdan, R. Marculescu, and S. Jain, "Dynamic power management for multidomain system-on-chip platforms: An optimal control approach," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, pp. 46:1–46:20, Oct. 2013.

[9] E. Seevinck, F. J. List, and J. Lohstroh, "Static-noise margin analysis of mos sram cells," in *IEEE Journal of Solid-State Circuits*, pp. 748–754, 1987.

[10] N. Verma and A. P. Chandrakasan, "A 65nm 8t sub-vt sram employing sense-amplifier redundancy," in *IEEE International Solid-State Circuits Conference*, pp. 328–606, 2007.

[11] N. Verma and A. P. Chandrakasan, "A 256 kb 65 nm 8t subthreshold sram employing sense-amplifier redundancy," in *IEEE Journal of Solid-State Circuits*, pp. 141–149, 2008.

[12] M.-H. Tu, J.-Y. Lin, M.-C. Tsai, C.-Y. Lu, Y.-J. Lin, M.-H. Wang, H.-S. Huang, K.-D. Lee, W.-C. W. Shih, S.-J. Jou, and C.-T. Chuang, "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," in *IEEE Journal of Solid-State Circuits*, pp. 1469–1482, 2012.

[13] I. J. Chang, J.-J. Kim, S. P. Park, and K. Roy, "A 32kb 10t subthreshold sram array with bit-interleaving and differential read scheme in 90nm cmos," in *IEEE International Solid-State Circuits Conference*, pp. 388–622, 2008.

[14] 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," in *IEEE Journal of Solid-State Circuits*, pp. 152–160, 2006.

[15] T. Suzuki, Y. Yamagami, I. Hatanaka, A. Shibayama, H. Akamatsu, and H. Yamauchi, "Fault-containment in cache memories for tmr redundant processor systems," in *IEEE Transactions on Computers*, pp. 386–397, 2002.

[16] G. Dhiman and T. S. Rosing, "Dynamic voltage frequecy scaling for multi-tasking systems using online learning," in *Proceedings of the International Symposium on Low Power Electronics and Desing*, 2007.

[17] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," in *Computer Architecture News*, 2005.

[18] J. L. Henning, "Spec cpu2006 benchmark descriptions," in *Computer Architecture News*, 2006.

[19] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, (Washington, DC, USA), pp. 65–76, IEEE Computer Society, 2010.

[20] 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 *Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI*, GLSVLSI '10, (New York, NY, USA), pp. 311–316, ACM, 2010.

[21] Y. Nakata, S. Okumura, H. Kawaguchi, and M. Yoshimoto, "0.5vv operation variation-aware word-enhancing cache architecture using 7t/14t hybrid sram," in *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '10, (New York, NY, USA), pp. 219–224, ACM, 2010.