Performance and Power Optimizations for Highly Reliable Caches

by

Seyedmostafa Azizabadifarahani
B.Sc. Ferdowsi University of Mashad, 2007
M.Sc. Shahid Beheshti University of Tehran, 2010

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

Supervisory Committee

Performance and Power Optimizations for Highly Reliable Caches

by

Seyedmostafa Azizabadifarahani
B.Sc. Ferdowsi University of Mashad, 2007
M.Sc. Shahid Beheshti University of Tehran, 2010

Supervisory Committee

Dr. Amiarali Baniasadi (Department of Electrical and Computer engineering)
Supervisor

Dr. Mihai Sima (Department of Electrical and Computer engineering)
Departmental Member

Abstract

Supervisory Committee
Dr. Amiarali Baniasadi (Department of Electrical and Computer engineering)
Supervisor
Dr. Mihai Sima (Department of Electrical and Computer engineering)
Departmental Member

This thesis introduces performance and power optimization techniques for caches. Our optimization techniques target both conventional caches, which are implemented using six-transistor (6T) cells, and highly reliable caches implemented using eight-transistor (8T) cells.

In 6T cell caches, we enhance leakage power dissipation by adapting a previous proposed technique, Drowsy Cache, according to the application behavior. We show that spatial locality in embedded applications is low and Drowsy Cache misses a significant leakage power saving opportunities. By taking a finer granularity approach, we achieve a significant leakage power reduction with minimal performance overhead.

Although 6T cell caches are commonly used, we show that they are not proper choice for future designs due to poor stability. We investigate 8T cells as alternative reliable designs for implementing caches. However, Column Selection Issue limits efficiency of 8T cells during write operations. Previous solution, Read-Modify-Write (RMW), addressed column selection issue by requiring a read operation before each write operation, imposing significant overhead on performance, cache traffic, and power.

We observe that a significant share of cache accesses in RMW is either redundant or unnecessary, consequently can be avoided without compromising program execution consistency. Based on our observations, we propose two techniques which exploit a

buffering mechanism to detect and filter out unnecessary and redundant cache accesses. Our simulation results show that our techniques improve performance and cache traffic effectively in 8T cell caches.

Furthermore, we propose a novel dual threshold 8T cell which reduces leakage power significantly with negligible impact on performance. Our proposed cell also improves stability and robustness to process variations compared to the conventional 8T cells.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I am grateful to my academic supervisor, Dr. Amirali Baniasadi, for his invaluable support, guidance, and friendship throughout my studies at UVic.

I would also like to thank my committee members, Dr. Mihai Sima and Dr. Yvonne Coady, for their insightful suggestions and comments.

I would like to thank Dr. Nikitas Dimopolus for his support and helpful feedbacks during our weekly group meetings.

Special thanks to all my friends at UVic (you know who you are) for all the time we spent and had fun in the lab and mostly around the foosball table.

Last but not least, I wish to thank my family for their never-ending source of support and encouragement.

Chapter 1

Introduction

Aggressive technology scaling has resulted in a gap between processor and memory speed [1]. This gap is expected to grow in future technology further. To bridge this speed gap, caches have been used traditionally to reduce the average memory access latency by capturing a small but popular fraction of applications' working set in a small and fast memory structures. Clearly, larger cache size can capture a bigger fraction of applications' working set and consequently improve the average memory latency [1]. However, power dissipation has become a limiting factor in design of larger caches. It has been shown that caches account for a big share of overall processor power dissipation [1].

Cache power dissipation can be divided into two main components: dynamic and leakage. Dynamic power refers to the power dissipated during cache accesses due to transistor activities in the internal cache structures. Hence, the higher number of cache accesses the higher dynamic power dissipation. On the other hand, leakage power dissipates in cache portions which have no activity. Leakage power stems from subthreshold leakage current in transistors and increases exponentially by reducing threshold voltage. Leakage power also has a proportional to the number of transistors and increases by the size of cache accordingly.

Many researches have focused on reducing dynamic or leakage power dissipations [2-12]. However, a popular technique to reduce power dissipation is supply voltage scaling. Both dynamic and leakage power benefit from supply voltage reduction, however, at the cost of an increase in cache access latency.

Cell stability is another design concern in caches especially in future technology nodes. By reducing supply voltage memory cells' state can change with a small noise level. Hence, designing stable cells which tolerate a high level of noise is preferable. Traditionally, caches are implemented using six-transistor (simply 6T) cells. 6T cells are area efficient, however, they can suffer from cell instability which limits voltage scaling. It has been also shown that 6T cells are very sensitive to process variations [29]. To address the instability issue many researches proposed several cells which improve cell stability at the cost of area overhead [13, 22, 23, 25].

By taking all these limitations into account, addressing all aforementioned requirements demands considering and revisiting caches from different perspectives and at different design levels including circuit and micro-architectural levels.

1.1 Contributions

The goal of this thesis is to optimize and improve L1 data caches from different perspectives including performance, power dissipation (both dynamic and leakage), and more importantly stability. In summary we make the following contributions:

- We study and categorize cache leakage power reduction techniques. Later, we focus on drowsy cache [2] which reduces leakage power by using two power supply voltages. We also show drowsy cache inefficiencies in applications with low spatial locality.
- We revisit drowsy cache and suggest that using drowsy cache at a finer granularity can improve leakage power further with a negligible impact on performance [48].

- We show that conventional six-transistor (known as 6T) cells are not the best choice for future cache designs due to poor stability. Hence, we focus on alternative cells, eight-transistor (simply called 8T) cells [24] which enhance stability by exploiting two extra transistors. We present that 8T cells result in another issue called column selection which is addressed previously by Read-Modify-Write (RMW) [29, 33] at the cost of extra cache accesses, limiting 8T cells effectiveness.
- We show that the overhead imposed by RMW is unaffordable and can be improved by considering application's behaviour and characteristics [41].
- We proposed two solutions, Write Grouping (WG) and Write Grouping and Read Bypassing (WG+RB) [41], to reduce RMW's overhead. These techniques exploit a novel buffering mechanism which detects and avoids unnecessary and redundant cache accesses. Our proposed techniques effectively enhance performance and cache traffic (which results in dynamic power reduction) with an affordable area overhead.
- We study sensitivity of performance to read and write latencies and find that a modest increase in write latency does not harm performance dramatically. Accordingly, we propose a dual threshold 8T cell which enhances leakage power dissipation with a slight increase only in write latency. We evaluate the influence of an increase in write latency on performance under our proposed techniques, WG and WG+RB, as well. We show that our proposed cell improves leakage power and cell stability effectively with negligible performance overhead.

1.2 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we review leakage power reduction techniques. We also present our proposed techniques to enhance the effectiveness of drowsy cache. In Chapter 3, we discuss cell stability in 6T cells compared to 8T cells. We also examine column selection issue in 8T cells and RMW as a solution. Moreover, we introduce two techniques (WG and WG+RB) to reduce RMW's overhead. In Chapter 4, we propose a dual threshold 8T cell and we evaluate its impact on leakage power, stability, and performance. Finally, Chapter 5 concludes and suggests future work.

Chapter 2

Application Specific Low Leakage Data Cache for Embedded Processors

2.1 Introduction

Power dissipation has become a limiting factor in design of high performance processors. In mobile computing era, power dissipation is also a crucial concern as it determines the battery lifetime. Dynamic and static powers are two main components of power dissipation in processors [53]. Dynamic power dissipates during transistors switching activities due to charging and discharging parasitic capacitances. Equation 2.1 shows the dynamic power dissipation formula, where C is parasitic capacitances, V_{DD} is supply voltage, and f is frequency of operations.

$$P_{Dynamic} \approx C.V_{DD}^2.f \qquad 2.1$$

As it is shown, by increasing frequency (performance) dynamic power increases almost linearly. Scaling supply voltage can enhance dynamic power dissipation effectively. On the other hand, scaling supply voltage reduces transistor switching speed which consequently imposes performance penalty [53]. Traditionally, threshold voltage of transistors is reduced to maintain performance. While reducing threshold voltage improves performance, it has a negative impact on static power dissipation.

Static power results from four leakage currents [53]: (1) reverse-biased junction leakage current, (2) gate induced drain leakage, (3) gate direct-tunneling leakage, and (4)

subthreshold (weak inversion) leakage. Among the aforementioned leakage current sources, subthreshold leakage is much larger than other leakage current components. Static power dissipation is formulated in the simplified form of Equation 2.2.

$$P_{Static} \approx I_{sub} V_{DD} \approx I_{D0} e^{\frac{V_{GS} - V_{th}}{nV_T}} V_{DD} \qquad 2.2$$

Where I_{sub} is subthreshold leakage current, V_{DD} is power supply voltage, and V_{th} is transistor threshold voltage. As this equation shows, static power has a linear relation with subthreshold leakage current and power supply. Note that subthreshold leakage current itself increases exponentially by reducing V_{th}. The traditional trend in voltage scaling [34] (to enhance dynamic power dissipation) and the following threshold voltage reduction (to maintain performance) increases the leakage power share by each technology downscaling.

Leakage current has a proportional relation with the number of transistors. Therefore, the components with higher number of transistors are likely to dissipate more leakage power. Caches account for a large share of chip transistors budget and it has been shown that a considerable share of overall processor's leakage power dissipates in caches. Previous studies have suggested solutions for reducing cache leakage power at circuit and micro-architectural levels [2-12].

The rest of this chapter is organized as follows. In Section 2.2, we briefly review the proposed techniques to reduce leakage power in caches. We focus on a low leakage power technique, known as Drowsy cache, and present its inefficiency in Section 2.3. We present our motivations in Section 2.4. In Section 2.5, we propose our techniques and in

Section 2.6 we demonstrate our simulation results. Finally, we offer concluding remarks in Section 2.7.

2.2 Cache Leakage Power Reduction Techniques

By considering Equation 2.2, static power reduces by scaling supply voltage or decreasing subthreshold current. As stated earlier, subthreshold current reduces exponentially by increasing threshold voltage [51]. Hence, the proposed cache leakage power reduction techniques can be divided into two main categories. The first approach aims at saving leakage power by turning off or scaling power supply. The second perspective focuses on reducing leakage power by exploiting high threshold transistors. While both approaches are very effective in reducing leakage power, they impose a considerable performance overhead.

2.2.1 Power Gating or Voltage Scaling Techniques

Turning off the power supply enhances leakage power effectively. Powell et al. [8] proposed cell power gating to reduce leakage current by exploiting a gating transistor. In this technique, the gating transistor can be either an NMOS transistor inserted between ground and the SRAM pull-down transistor or a PMOS transistor inserted between the power supply and the SRAM PMOS transistor. By turning off the gating transistor and moving cells into the low leakage mode (here referred to as sleep mode), the leakage current is reduced significantly. To reduce the area overhead, all cells in a specific cache portion (e.g., a cache line) share the same gating transistor. Therefore, by exploiting one gating transistor significant leakage power reduction can be achieved. However, the disadvantage of this technique is losing stored data as cells move into the sleep mode.

Consequently, the following references to the cache lines in the sleep mode incur cache misses resulting in performance slowdown.

Agrawel et al. [21] showed that it is possible to keep data even after turning off the power supply, retrieving the stored data. They proposed to control gating transistors using the word line of each cache row. In this way, all cells in one cache row turn on upon a cache access and switch back to the sleep mode thereafter. As cell stability decreases rapidly by turning off cell power supply, several techniques [2, 9, 10] took a more conservative approach and assumed that the stored data is lost after turning off the cell power supply.

Hence, several researches [2, 9, 10, 11, 15, 17] focused on designing efficient techniques to select the best cache line candidates for sleep mode. Ideally, the cache lines that are no longer needed (dead cache line [54]) are the best choices to be turned off. Because they do not contribute to performance and will be evicted eventually upon a cache replacement. [17] demonstrated the correlation between the spatial pattern accesses with the instruction addresses and data references offsets within a cache line. By taking this correlation into account and exploiting a direct map array, they proposed a spatial pattern predictor. They predict the pattern of access to the cache lines and force moving some cache lines into the sleep mode as they are not likely to be accessed.

Kaxiras et al. [9] studied application behavior and proposed time-based policies that turn off cache lines after a certain number of cycles assuming the cache lines are not likely to be accessed again. Their time-based policies need to be adjusted and tuned for each application. They also proposed a dynamic technique aiming at finding the optimum number of cycles at run time according to the application behavior.

Zhou et al. [15] suggested turning off only the data array while keeping the tag array in active mode. This enables monitoring cache accesses and estimating the hypothetical miss rate if all cache lines were in the active mode. By having this information, the hardware adjusts itself to the application behavior to minimize miss rate.

Kim et al. [2] proposed circuit solutions to design caches equipped with an extra mode, referred as drowsy mode. In this technique, cells can operate either in active mode (regular supply voltage level) or in drowsy mode (a supply voltage lower than regular power supply). Cells operating in the active mode are accessible while dissipating considerable leakage power. On the other hand, cells in the drowsy mode save leakage current due to lower supply voltage level without losing the stored data. However, cells in the drowsy mode are not accessible and need to switch back to the active mode imposing an extra latency. They also suggested microarchitectural mechanisms to reduce the performance overhead associated with drowsy caches [12]. In order to bring back cache lines to the drowsy mode, they considered two policies, simple and noaccess. In simple policy, they put all cache lines in the drowsy mode at the end of fixed intervals. In noaccess policy, they keep track of accesses made to cache lines and put those cache lines in drowsy mode which were not accessed during the previous fixed intervals.

Shawkey et al. [11] proposed an aggressive drowsy cache. They proposed a circuit that keeps all cells in drowsy mode and switches the desired cells back to the active mode only upon read or write operations. Their proposed moving back cells into drowsy mode immediately after completing the read or write operations.

2.2.2 Dual Threshold Voltage Techniques

Increasing threshold voltage improves leakage power exponentially. Accordingly, one can suggest using high threshold transistors with the aim of improving leakage power. However, the associated performance overhead due to the increase in threshold voltage can limit the effectiveness of this technique. Hence, several researches [4-7] focused on exploiting selectively high threshold transistors along with regular threshold transistors to maximize leakage power saving with minimal performance overhead.

Kim et al. [4] proposed exploiting dynamically controlled threshold voltages to reduce the leakage power in SRAMs. They used forward body-biasing to control the threshold voltage of transistors of each cache line separately. In order to minimize the energy and delay overhead, a cache line is switched to high V_{th} when deemed unlikely to be accessed anymore. Kim et al. [5] presented a forward body-biasing technique for leakage power reduction in cache memories by including device-level optimizations into circuit-level techniques. They utilized super high V_{th} devices to suppress the leakage power in unselected portions of a cache. Meantime, performance is maintained by using dynamic forward body-biasing in the selected SRAM cells.

Azizi et al. [6] presented a family of asymmetric low leakage SRAM cell designs. Their solution relies on the observation that in typical applications most of the bits in caches are zeroes for both the data and instruction streams [16]. Their proposed cells dissipate less leakage power in the frequent (zero) state. On the negative side, exploiting high threshold transistors asymmetrically has a negative impact on cell stability and latency. They also proposed a novel sense amplifier that mitigates the increase in read latency.

Amelifard et al. [7] proposed several symmetric dual threshold cells with different leakage and latency characteristics. Their method is based on the observation that the read and write delays of a memory cell in an SRAM block depend on the physical distance of the cell from the sense amplifier and decoder. They proposed an algorithm that determines the configuration for each cell based on its location, given a timing constraint. Their algorithm employs slow cells (low leakage) in locations close to decoders and sense amplifiers and fast cells (high leakage) in far locations. As a result, they balanced leakage power dissipation and performance requirements by considering physical location of cells. Lee et al. [40] extended [7] and suggested a similar algorithm that considered cell stability along with leakage and latency characteristics.

2.3 Drowsy Cache

In this section, we look at drowsy cache [2] in more detail. In Figure 2.1 we present an SRAM cell equipped with two different voltage modes: V_{DD} (normal) and V_{DDLow} (drowsy). The cell uses complementary control signals referred to as Low_{Volt} and its inverted version to decide the voltage connected to the cell. Changing the cell's mode from drowsy to normal comes with a latency overhead required to restore the voltage level of the node from V_{DDLow} to V_{DD}, which is referred to as wakeup latency.

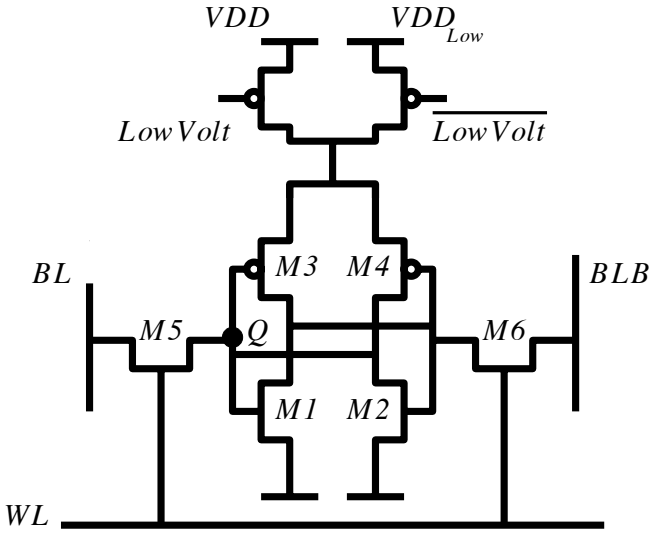

Figure 2.1 - A drowsy SRAM cell with the supply voltage control mechanism

Figure 2.2 shows the required circuits to implement drowsy cache. The word-line gating circuit is used to prevent cache line access when it is in the drowsy mode. Whenever a cache line is accessed, the cache controller monitors the condition of the voltage of the cache line by reading the drowsy bit. If the accessed cache line is in the active mode, we can read out the contents of the cache line without performance loss. No performance penalty is incurred, since power mode is probed by reading the drowsy bit in parallel to reading the data and tag comparison. In the event when the cache line is in the drowsy mode, we prevent reading the content as it may result in reading out the incorrect data. Under such circumstances the cache line is waken up automatically during the next cycle(s), and the data could be read afterwards.

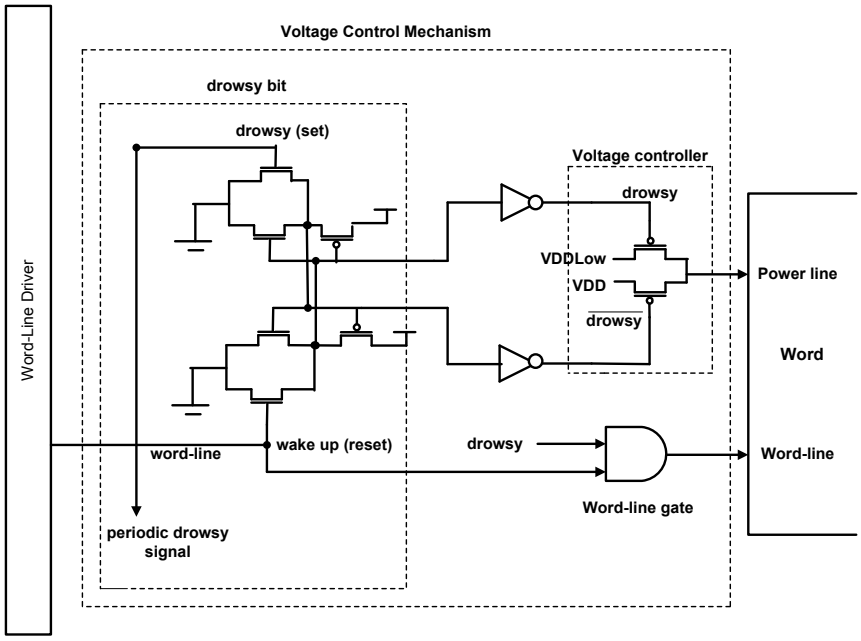

Figure 2.2 - Drowsy cache implementation

In drowsy cache [2], two policies (*simple* and *noaccess*) have been introduced for deciding when and which cache lines move to the drowsy mode. In both policies, the decisions are made periodically at certain intervals referred to as Update Window (or simply UW). The UW size has a significant impact on performance and power. In *simple* policy, all cache lines are moved into drowsy mode at the end of UWs. However, in *noaccess* policy only cache lines that have not been accessed in the previous UW are moved into drowsy mode. In *noaccess* policy, the wakeup latency overhead is reduced compared to simple policy at the cost of losing leakage power saving opportunities.

2.4 Motivation

Drowsy caches take a non-discriminating approach in dealing with words in a cache line, moving the entire cache line into and out of the drowsy mode. This relies on the assumption of high spatial locality and that all words in a cache line are needed at the

same time. This results in keeping a cache line in high leakage current mode for long periods even if only a small subset of words in the cache line is needed. An undesirable consequence of this approach is losing frequent power reduction opportunities: many unused words have to stay in the high leakage mode without ever being used and just because they belong to a line where a single word is accessed.

On the contrary to the above assumption, our study shows that only a small share of the words in a cache line, are accessed during UWs. In other words, while some portions of the cache line are accessed frequently and regularly during an UW, there are parts, which are not accessed at all. To provide better insight, in Figure 2.3 we report the breakdown of how many words become active during different UW sizes in the data cache (see Section 2.6 for details). As presented, for example, more than 60% of the time, only one word is active during a 64-cycle UW size. Also, about 80% of the time, we only have one or two active words. As presented active word breakdown varies with UW size. This is due to the fact that exploiting larger UWs sizes could result in accessing a larger group of active words, effectively reducing the share of lines including only a single active word. Note that our study shows that variations in cache size or wakeup latency do not make a significant impact on active word breakdown.

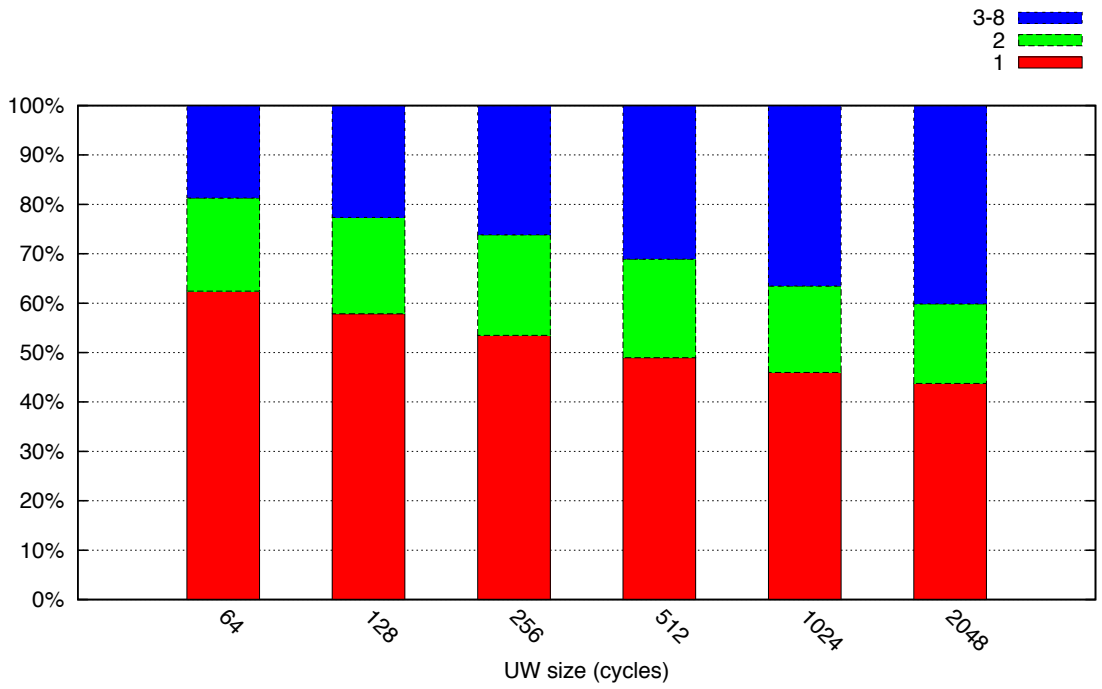

Figure 2.3 - Breakdown of the number of active words for different UW sizes

This observation motivates us in using drowsy cache in a finer granularity level to increase the leakage power saving opportunity. In this thesis, we investigate the word-size (eight bytes) granularity instead of cache line-size (64 bytes) granularity. Henceforward, we refer to the original drowsy caches as line-drowsy caches. By using a word-size granularity approach we reduce leakage power compared to line-drowsy caches as we put words in the drowsy mode which were previously kept in the high leakage mode just because they were in a cache line with some needed words.

As our optimizations rely on how predictable the behavior presented in Figure 2.3 is, we also study word activity predictability. In Figure 2.4 we report how often a word, which is active during the most recent *n* UWs, will also be active during the next UW. Bars from left to right report for *n* values equal to one, two and three, respectively. For

example the left bar (n=1) reports how often a word which is active in the current UW will be active in the next one. The middle/right bars show how often the word being active in the most recent two/three UWs will be active in the next one. As presented the likelihood of a word being active in n consecutive UWs reduces as n grows.

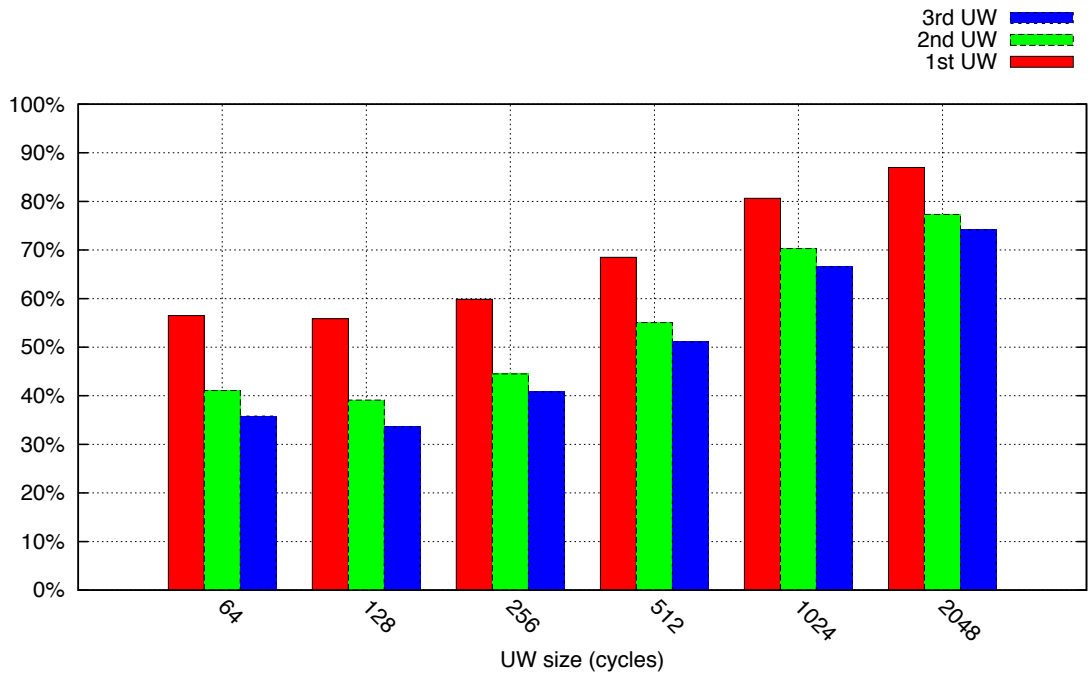

Figure 2.4 - Word activity predictability

2.5 Proposed Techniques

In this section, we introduce Application Specific Low Leakage Caches (ASL) to address the aforementioned inefficiency in line-drowsy caches. ASL reduces leakage power compared to line-drowsy caches as it moves unused words (ignored by line-drowsy caches) into the drowsy mode. Similar to line-drowsy caches, ASL picks the words moved in and out of the drowsy mode based on the application behavior.

We present two variations of ASL. In our first approach (similar to *simple* line-drowsy), we move cache words into the drowsy mode at the end of each UW. When a word is needed we only wake up that word and keep the rest of the cache line in drowsy mode. Consequently, we save energy for the inactive words without paying performance penalty. In the event that more than one word is needed we pay the wakeup performance penalty more than once hence could lose performance compared to line-drowsy caches. We refer to our first solution as the basic ASL or B-ASL.

In our second approach and to protect performance more aggressively (similar to *noaccess* line-drowsy), we speculatively identify active words and avoid moving them into the drowsy mode at the end of each UW. Consequently, while we still save energy for the drowsy words, we maintain performance by keeping the active words awake. We refer to this variation as performance-aware ASL or P-ASL. In P-ASL we use a single status-bit per word to record word activity. This bit is set to one when the word is accessed. At the end of a UW all words are moved into the drowsy mode with the exception of those with the status-bit set to one. The status-bit is returned to zero at the end of the next UW to avoid keeping inactive words out of the drowsy mode for long times. P-ASL is motivated by our observation that words accessed during a recent UW are often accessed in the next UW as well (see Figure 2.4).

It is worth pointing out that B-ASL and P-ASL come with small overhead. Similar to the circuits used in line-drowsy caches (shown in Figure 2.2), we require memory cells to keep the word mode and pass transistors to pass the proper supply voltage to each word. ASL's overhead is different from line-drowsy as its associated overhead is applied per word and not per cache line. In line-drowsy for each cache line one drowsy circuit

(presented in Figure 2.2) is required. However, in ASL and for a cache with 64B line size, eight drowsy circuits are needed.

2.6 Evaluation

2.6.1 Methodology

We used Simplescalar [18], a cycle accurate simulator. Table 1 shows processor configuration. Also, we use a representative subset of MiBench benchmarks [19]. We run each simulation for 500M or up to completion, whichever comes first. For cache power estimation, we use CACTI [20].

Table 1 - Processor configuration

Processor Component	***Value***
Integer Functional Unit	4 ALU, 1 Multiplier/Divider
Floating-point Functional Unit	4 ALU, 1 Multiplier/Divider
Instruction Fetch Queue / LSQ / RUU size	4/64/32
Decode/Issue/Commit Width	4/4/4
Memory Latency	First Chunk 512 cycles Inter Chunk 64 cycles
Branch Predictor	Comb: 1024 meta size, Bimodal: 2048, 2level: 8bits history and 1024 array, BTB: 512, 4-way, 3 cycles mis-prediction penalty
Instruction Cache	128KB, 8-way, 3 cycles latency
Data Cache	32KB, 4-way, 3 cycles latency
Level 2 Cache	Unified, 2MB, 8-way, 32 cycles latency

In the following sections, we report our experimental results. We report data cache leakage power and performance for B-ASL and P-ASL compared to line-drowsy caches. As explained earlier, line-drowsy has two policies: *simple* and *noaccess*. In the interest of

fairness we compare B-ASL to the *simple* policy (whose only goal is leakage reduction) and P-ASL to the *noaccess* policy (which aims at protecting performance). We also report how variations in wakeup latency and UW size impacts results.

2.6.2 Data Cache Leakage Power

As stated, B-ASL takes a fine-grain approach to reduce leakage power dissipation by putting unused words into the drowsy mode. Figure 2.5 reports leakage power reduction for B-ASL compared to line-drowsy *simple* policy. As presented, on average, B-ASL improves leakage power by more than 88%.

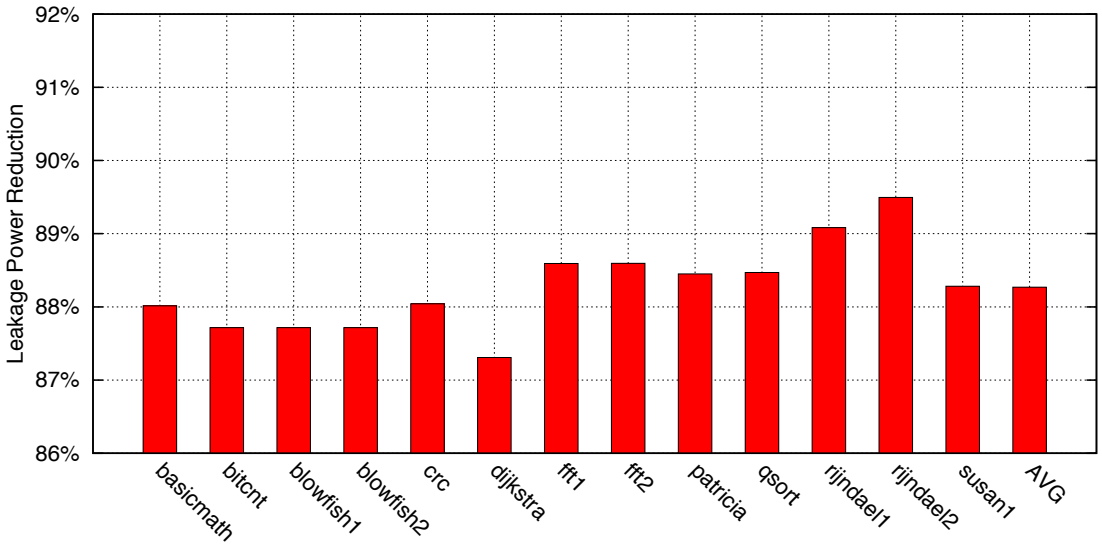

Figure 2.5 - B-ASL leakage power reduction compared to line-drowsy simple policy

Figure 2.6 shows leakage power reduction for P-ASL compared to line-drowsy noaccess policy. As presented, we witness around 89% leakage power reduction.

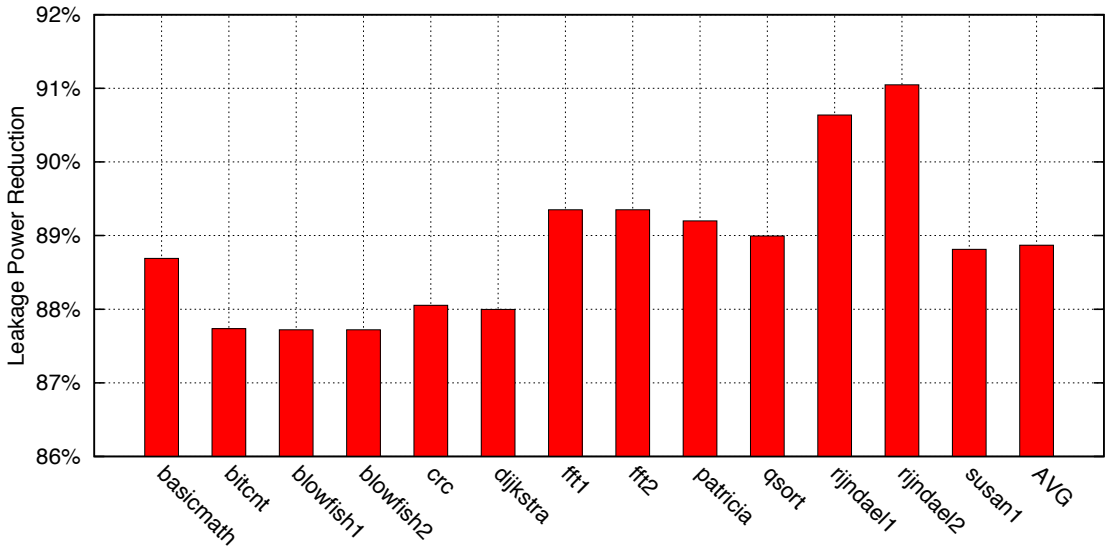

Figure 2.6 - P-ASL leakage power reduction compared to line-drowsy noaccess policy

2.6.3 Performance

Drowsy caches reduce leakage power dissipation. However, the extra wakeup latency required during cache access for drowsy cache lines and words could harm performance. In this section we report performance for our solutions compared to the counterpart line-drowsy caches.

As presented earlier, in ASL each word is awakened separately. In the event where more than one word in a drowsy cache line is accessed this would impose wakeup latency for every accessed word. However, as reported in Figure 2.3, only a small fraction of cache words are accessed during UWs. Consequently, we expect the performance loss associated with ASL to be negligible. This is validated by the results reported in Figure 2.7 and Figure 2.8. In Figure 2.7 we report performance for B-ASL compared to line-drowsy *simple* policy. As reported, while five of the applications show little or zero performance loss, average performance loss is 0.7%. *Dijkstra* shows the highest performance loss (3.1%).

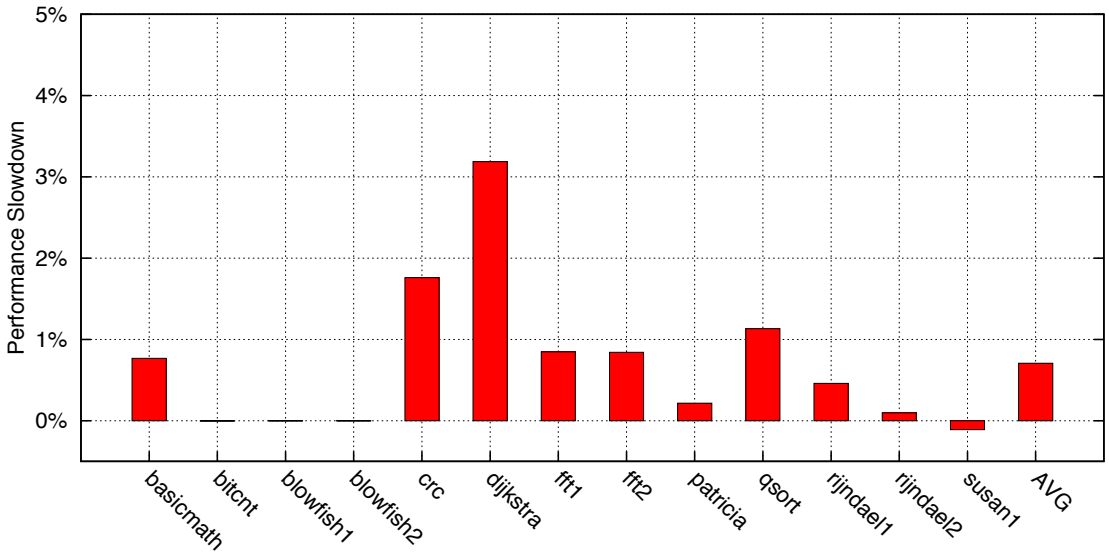

Figure 2.7 - B-ASL performance slowdown compared to line-drowsy simple policy

Figure 2.8 shows performance loss for P-ASL compared to line-drowsy *noaccess* policy. On average, P-ASL shows a performance loss of 0.5%. Two applications, *crc* and *dijkstra*, experience the highest performance slowdown around 4.5% and 3.5%, respectively.

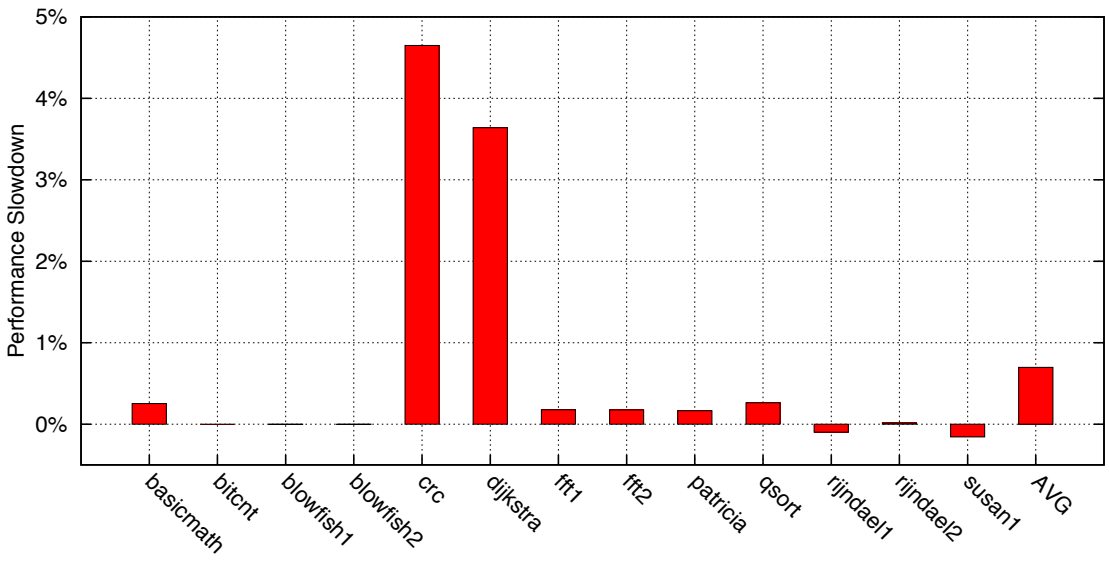

Figure 2.8 - P-ASL performance slowdown compared to line-drowsy noaccess policy

2.6.4 Sensitivity Analysis

In this section we report how variations in wakeup latency and UW size impacts average leakage power dissipation and average performance for the studied applications.

2.6.4.1 Wakeup Latency

Switching between two power supplies could not be accomplished immediately. The time overhead associated with this switching can vary from one technology to another. To provide better understanding, we vary wakeup latency from two to four cycles and report average performance slowdown compared to a system with one cycle wakeup latency. Figure 2.9 shows that wakeup latency has a significant impact on performance. As presented, increasing wakeup latency reduces average performance in all cache systems. Among the four methods studied here B-ASL and P-ASL are more sensitive to wakeup latency. This could be explained by the fact that ASL suffers from additional

wakeup latencies compared to line-drowsy when more than one word is accessed in a cache line.

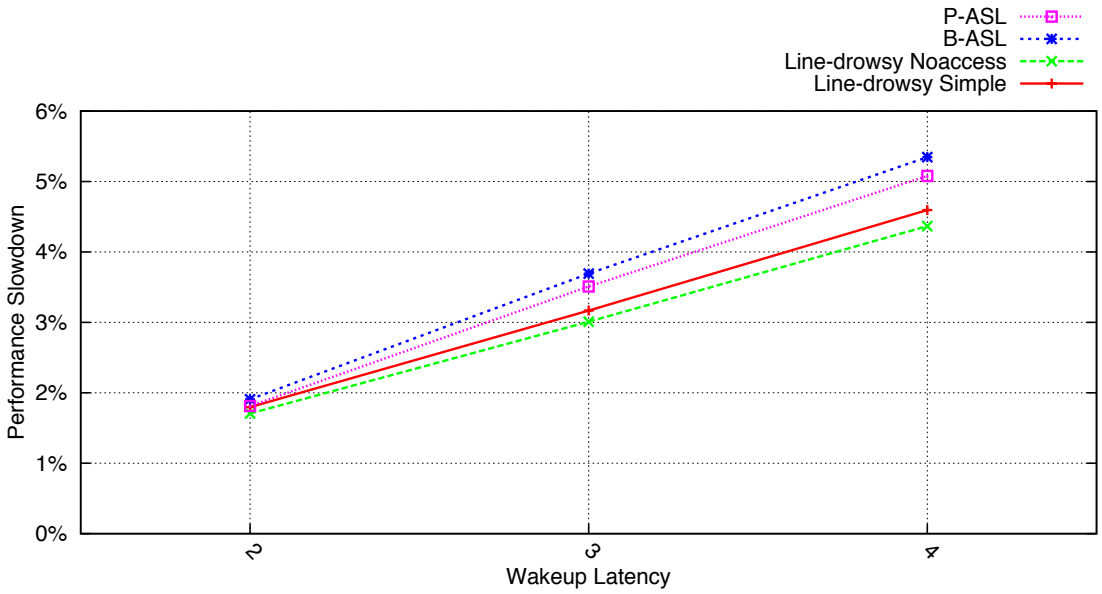

Figure 2.9 - Wakeup latency impact on performance

Longer wakeup latency could increase execution time (and hurt performance) and consequently increase leakage power dissipation. Figure 2.10 shows average leakage power dissipation for wakeup latencies of two, three, and four cycles relative to a system using one cycle wakeup latency. As this figure shows, relative power dissipation in two ASL variations is more sensitive to wakeup latency compared to line-drowsy policies.

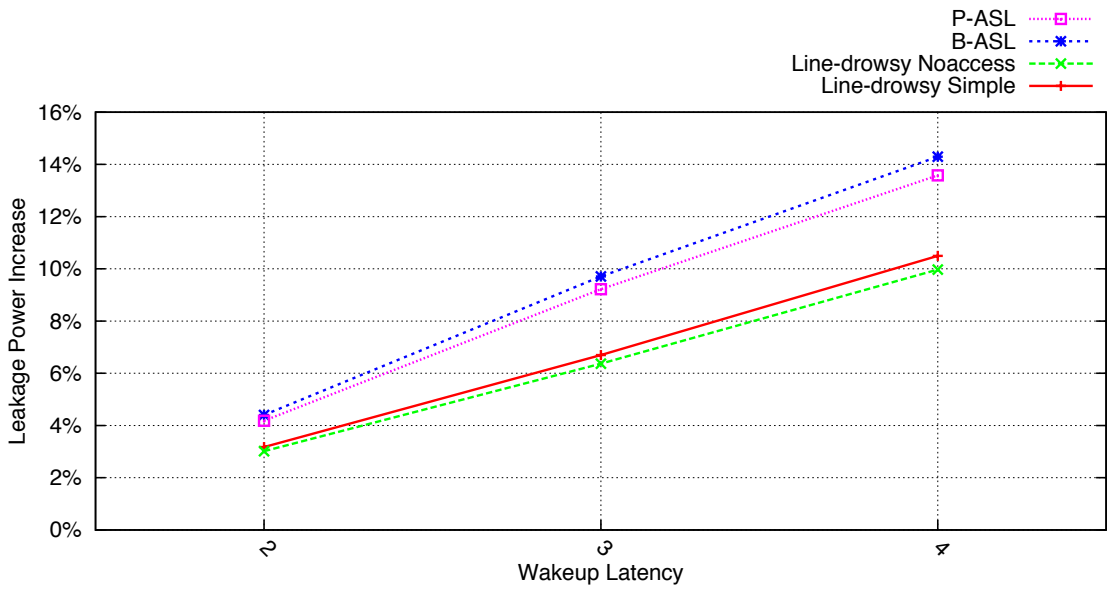

Figure 2.10 - Wakeup latency impact on leakage power dissipation

2.6.4.2 Update Window Size

In this section, we investigate how UW size impacts performance and power. We vary UW size from 256 to 32768 cycles and report average performance improvement compared to a 128-cycle UW size. Figure 2.11 shows average performance improvement for the cache systems investigated in this work. As presented, increasing UW size improves performance. This could be explained by the fact that increasing UW size results in moving the cache lines or words into the drowsy mode less frequently. It is worth pointing out that for UW sizes higher than 4096, ASL shows better performance improvement compared to line-drowsy.

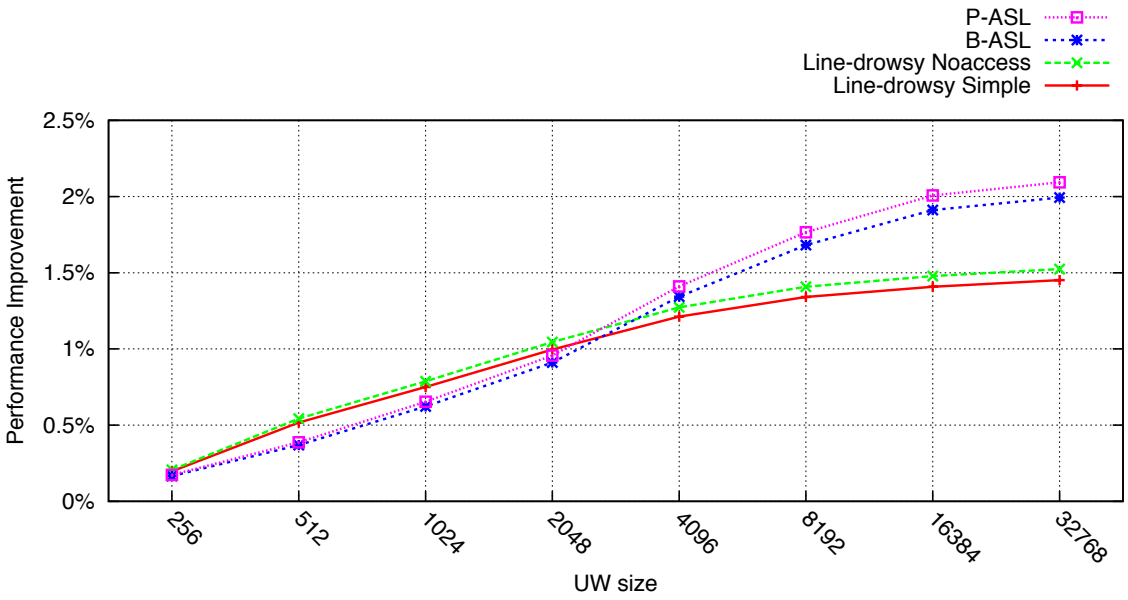

Figure 2.11 - UW size impact on performance

In Figure 2.12 we report increases in average leakage power dissipation for different UW sizes compared to a 128-cycle UW size. As presented, leakage power dissipation increases with UW size. Increasing UW size results in less frequent cache lines and words moved into the drowsy mode. Hence power dissipation increases.

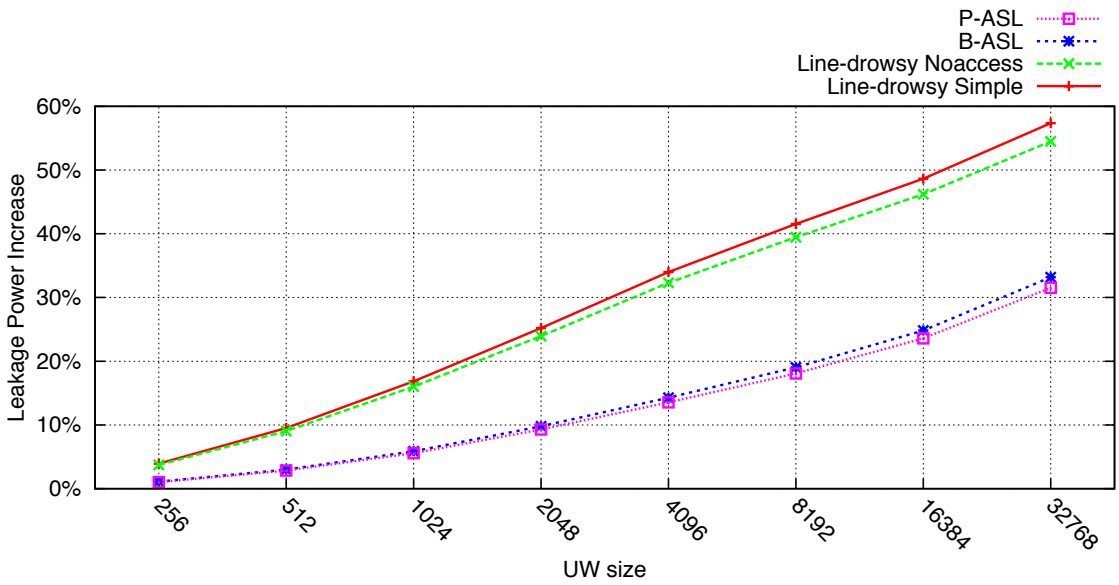

Figure 2.12 - UW size impact on leakage power dissipation

2.7 Conclusion

In this chapter, we presented Application Specific Low Leakage Data Caches. Our solutions revisit drowsy caches by taking a word-size granularity approach in moving words in and out of the drowsy mode. We are motivated by two observations. First more than 80% of the time only one or two words in a cache line are accessed during a UW. Second, activated words during a UW are often reactivated during the next UW. By using our solutions we reduce data cache leakage power significantly compared to previously suggested solutions. We reduce leakage power while maintaining performance at a competitive level with alternative low leakage caches.

Chapter 3

Column Selection Solutions for L1 Data Caches Implemented using 8T Cells

3.1 Introduction

In Chapter 2, we discussed power dissipation and how it has become a challenging issue in design of mobile and high performance computing devices. We also explained that voltage scaling is a common technique to reduce power dissipation. On the negative side, reducing voltage can result in an increase in transistor switching latency, which in turn requires a longer time to operate correctly.

To balance between power and performance requirements, Dynamic Voltage and Frequency Scaling (DVFS) has been introduced to scale voltage and frequency adaptively [35-37]. DVFS is able to deliver the required performance while minimizing power dissipation. DVFS switches between predefined voltage levels dynamically according to the required performance and power demand [37]. The more the number of voltage levels the higher the chances of operating at the optimal voltage and frequency levels. Among the different levels, the minimum voltage level V_{min}, assuring correct operation, limits the lowest operating voltage.

One of the system components likely to serve as the bottleneck in deciding V_{min} is the cache, which is traditionally implemented using six-transistor cells (referred to as 6T cells) [26, 28, 38]. 6T cells, however, suffer from poor stability under technology scaling. Cell stability in 6T cell is also aggravated more by voltage scaling [33].

To overcome the stability issue, Chang et al. [24, 33] introduced Eight-transistor Cells (or simply 8T cells). In 8T cells, stability is improved by exploiting extra transistors, allowing scaling supply voltage even below the threshold voltage [27]. Despite the achieved cell stability in 8T cells, caches implemented using 8T cells suffer from column selection issue due to bit interleaved SRAM arrays. Bit interleaving is a common technique used to avoid soft errors and prevent multi-bit upsets in one word [39]. From now on, we refer to caches implemented using 6T cells as 6T cell caches. Similarly, 8T cell caches refers to the caches implemented using 8T cells.

Morita et al. [33] studied the column selection issue in SRAM arrays using 8T cells. They proposed the Read-Modify-Write (also known as writeback) scheme to solve this issue. In Read-Modify-Write (RMW), the addressed row in an SRAM array has to be read and buffered before being written. This imposes an extra read for every write operation. The extra read per each write operation required by RMW has a negative impact on performance and cache traffic (and consequently power dissipation).

In this thesis, we identify and present the inefficiency of RMW. We show that a considerable share of cache traffic overhead imposed by RMW is either redundant or unnecessary and consequently can be eliminated without compromising coherency and correctness concerns [41]. We propose two solutions, Write Grouping (WG) and Write Grouping and Read Bypassing (WG+RB) to reduce the overhead associated with RMW. We evaluate the efficiency of WG and WG+RB in cache access frequency reduction and performance improvement.

The rest of this chapter is organized as follows. In Section 3.2, we discuss SRAM cells and compare 6T cells with 8T cells. We also briefly talk about column selection issue and

the proposed techniques to address it. In Section 3.3, we present inefficiencies in previous proposed techniques and our motivations to improve them. In Section 3.4 we propose our solutions and in Section 3.5 we evaluate our proposed techniques. Finally, we offer concluding remarks in Section 3.6.

3.2 Background

SRAM arrays are implemented as a matrix of Memory Cells (MCs) surrounded by the peripheral circuits as shown in Figure 3.1. MCs are arranged in a two-dimensional array. Decoders on the left side of matrix drive word lines to select a row in the matrix. Word lines are shared among MCs in one row. The precharge circuit shown on the top figure controls and manages precharging the bit lines. As depicted, bit lines are shared among MCs in one column. The sense amplifiers are used to speed up voltage sensing on bit lines to enhance read latency. At the bottom of the figure, the column multiplexers decide the connection of bit lines to the outputs.

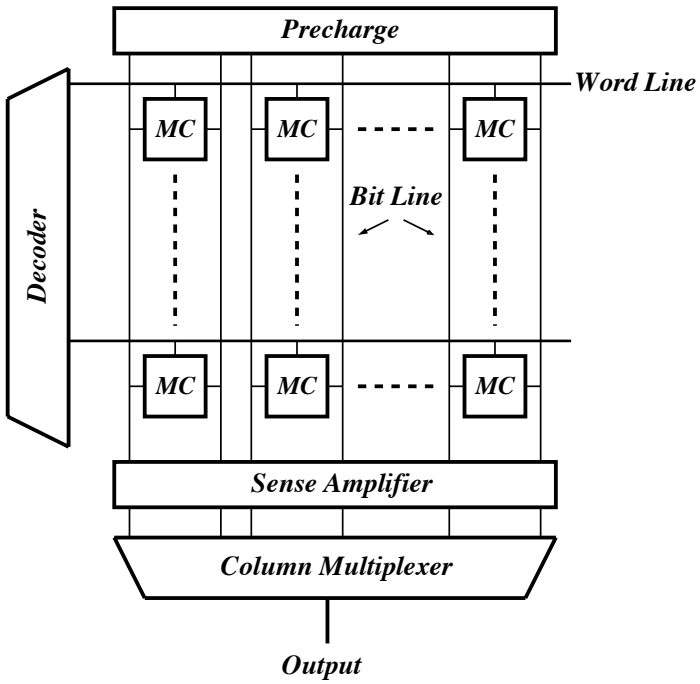

Figure 3.1 - SRAM array architecture

3.2.1 Six-transistor SRAM Cells

Conventionally, MCs in on-chip caches are implemented using six-transistor SRAM cells. We show a 6T cell in Figure 3.2. As this figure shows, four transistors (M1 through M4) form two cross-coupled inverters (simply referred to as the storage node). This storage node can keep two stable states (i.e., one and zero). Two access transistors (M5 and M6), two bit lines (BL, BLB), and one word line (WL) are used to perform read and write operations.

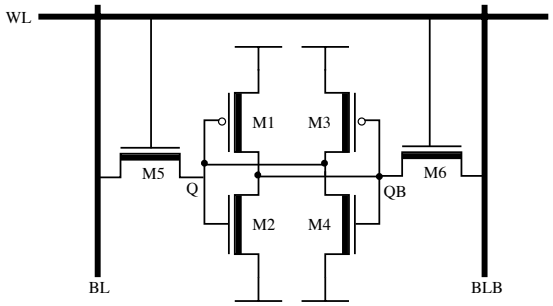

Figure 3.2 - A 6T Cell

3.2.1.1 Read Operation

Figure 3.3 shows a 6T cell during a read operation assuming the cell content is initially one (*Q*=1, *QB*=0). By taking the following steps, a read operation is performed:

1) Initially, both bit lines (BL and BLB) are precharged to one by precharge circuit.

2) WL rises to initiate the read operation.

3) The access transistor, M6 in this example, turns on and discharges the bit line.

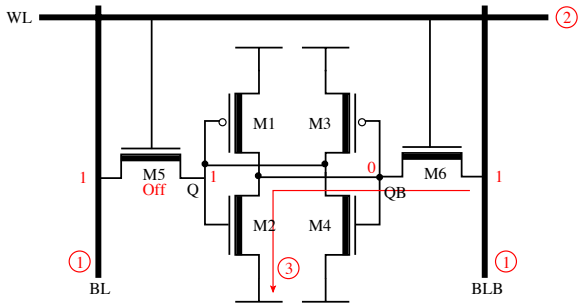

Figure 3.3 - Read operation in 6T cells

In this figure, BLB discharges through M6 and M2. During the discharge phase, *QB*'s voltage increases to a positive level due to the voltage division across M6 and M2. If this level exceeds the switching threshold of the other inverter, the cell flips and data is lost affecting read stability. Read stability indicates how stable the cell is during read

operations. To enhance read stability, often a pull down network stronger than the access transistors is used.

At the end of bit lines (not shown in Figure 3.3), the sense amplifier is exploited to speedup read operations. Sense amplifier detects the voltage difference between bit lines at the early stage of bit line discharge.

3.2.1.2 Write Operation

In Figure 3.4 we show a 6T cell during a write operation. We assume that the current content of cell is one (Q=1, QB=0) and the new value to be written is zero. For write operations the following steps are taken (Note that the bit lines are precharged to one initially):

1) The new value is loaded on BL and its complementary value is loaded on BLB.

2) WL rises to initiate a write operation.

3) If the current value is different from the new one, the access transistor on the side of the cell storing one (M5 in this example) is turned on. The internal node's (node Q) voltage drops, triggers the other inverter and finally flips the storage node.

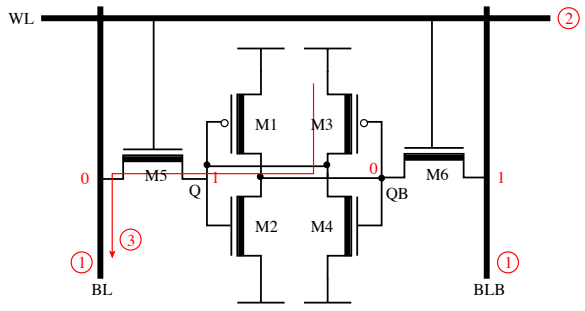

Figure 3.4 - Write operation in 6T cells

In this figure, M5 turns on and causes current flow through M3 and M5. Due to the voltage division across M3 and M5, *Q*'s voltage drops. *Q*'s voltage has to drop below the switching threshold of the other inverter (referred as write trip point) to flip the other inverter and store the new value. The higher the write trip point, the lower write latency and power dissipation. Access transistors stronger than the pull up network are required to improve the write trip point.

3.2.1.3 Stability Analysis

Static Noise Margin (SNM) refers to the minimum required DC voltage to flip cell state. In other words, SNM shows cell stability in the presence of static noises. A balance between pull-down and pull-up network quarantines higher SNM level. This is opposing read stability and write trip point requirements discussed earlier. As we showed, read stability improves by using a pull down network stronger than access transistors. In addition, proper write trip point is achieved by employing a pull up network stronger than access transistors. These requirements impose conflicting design constrains which make designing reliable and stable SRAM cells very difficult.

Furthermore, cell stability in 6T cells is very sensitive to process variation. A previous study [24] has shown that process variations such as dopant fluctuations and line edge roughness make Voltage Transfer Characteristic (VTC) of 6T cell asymmetric and consequently can hurt cell stability further. Moreover, due to threshold voltage variation, supply voltage scaling can further exacerbate the cell stability problem. To overcome cell stability concern in 6T cells, Chang et al. [24] proposed 8T cells.

3.2.2 Eight-Transistor SRAM cell

As shown in Figure 3.5, an 8T cell is composed of a 6T cell and two additional transistors (M7 and M8). Since read and write ports are separated in 8T cells, dedicated word lines and bit lines are required for read and write operations. Hence, Write Word Line (WWL) and Write Bit Lines (WBL and WBLB) are used during write operations while Read Word Line (RWL) and Read Bit Line (RBL) are needed for read operations. Write operations in 8T cells are similar to write operations in 6T cells.

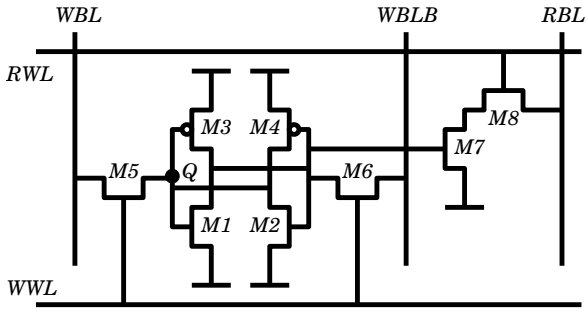

Figure 3.5 - An 8T cell

However, read operations in 8T cells are performed differently. Figure 3.6 shows the steps taken during a read operation in 8T cells:

1) Initially, RBL is precharged to one.

2) RWL rises to initiate the read operation.

3) If the cell value is zero (Q=0 and QB=1) M7 turns on and discharges RBL through M8. If the cell value is one (Q=1 and QB=0) M7 remains off and no bit line discharge is required.

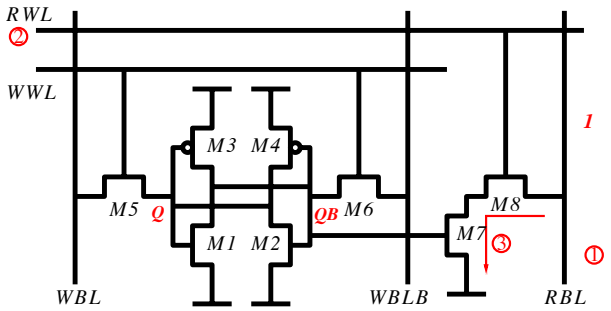

Figure 3.6 - Read operation in 8T cells

As read and write ports are separated in 8T cells, it is possible to optimize the cell for read stability and write ability simultaneously. Hence, the storage node can be optimized for write trip point and SNM constraints with no consideration for read stability. Moreover, it has been shown that 8T cells are more robust to process variations [29].

In addition to enhancing cell stability, 8T cells support multiport caches (one read and one write). Employing multiport caches improves processor performance due to allowing servicing multiple cache requests in parallel [31]. However, while multiport caches (implemented using 6T cells) come with significant area and power overhead [20], 8T cells inherently support one read and one write operations.

Moreover, Morita et al. [29] showed that 8T cells are more area efficient compared to 6T cells in technology nodes beyond 45nm even by accommodating two extra transistors and the associated word line and bit line.

3.2.2.1 Column Selection Issue

Bit interleaving is commonly used to reduce the probability of upsetting two bits in one word making using simple and low cost one bit correction techniques possible [39]. Accordingly, any two adjacent cells in one row (shown Figure 3.1) belong to two different words. Hence, an activated row by a word line selects all cells in one row, while

only a subset of cells is intended to be selected. Column selection refers to this issue. In this thesis, we refer to the cells in the columns that are intended to be selected as selected columns. The cells that are not intended to be selected are referred to as half-selected columns.

During read operations regardless of MC type (6T cell or 8T cell), all cells in the activated row experience a read operation, however, only the selected columns are routed to the output (decided by the column multiplexers shown in Figure 3.1). In other words, all cells in the selected columns and half-selected columns perform a read operation while only the selected columns are routed to the output.

In 6T cells caches, as we explained earlier, all bit lines are precharged before each write operation. Then, the write drivers load new values to the bit lines in the selected columns with no impact on the bit lines in the half-selected columns. Therefore, the bit lines in the half-selected columns keep their precharged state while the bit lines in the selected columns are loaded. Finally, by asserting one word line, the new values are written to the cells in the selected columns. Under this circumstance, the cells in the half-selected columns perform a dummy read operation which does not have any impact on the cells' content. In other words, while cells in the selected column are written, the cells in the half-selected columns perform a read operation.

Although the SRAM architecture of 8T cell caches is very similar to the 6T cell caches, a similar technique cannot be employed for 8T cells caches. This is due to the fact that 8T cells are optimized for write operations and biasing cells for the read operation can corrupt the stored data. Hence, the column selection needs to be addressed in 8T cells caches differently.

3.2.3 Related Work

Chang et al. [33] proposed to address the cells in one row at the word granularity level in non-bit interleaved SRAM arrays. Therefore, they were able to select only the desired cells in the selected row. This technique, however, requires multi-bit correction techniques and larger write word line drivers, which could increase area and power dissipation.

Morita et al. [24] proposed Read-Modify-Write (RMW) as a write mechanism in 8T cells caches. In RMW, a read operation is required before each write. In Figure 3.7 we show the sequence of steps needed in RMW. The block diagram at the bottom of the figure shows writeback and write driver circuits that are involved in write operations.

The sequence of steps for RMW is as follows:

1) Initially, precharge circuit charges RBLs of all columns.
2) Read word line driver raises the RWL to initiate a read operation. In this phase of RMW, multiplexers do not route data to the output.
3) After performing the read operation, latches at the bottom of columns store the data.
4) The multiplexers controlled by *Write-back* signal load write drivers. Write drivers in the selected columns are loaded by *Data-in* path while write drivers in the half-selected columns are loaded by latches. Then, write drivers drive write bit lines (WBLs and WBLBs) with new values.
5) The write operation is finalized by raising the WWL and writing the value on write bit lines to the cell. This operation (writing back to the SRAM row) is called writeback.

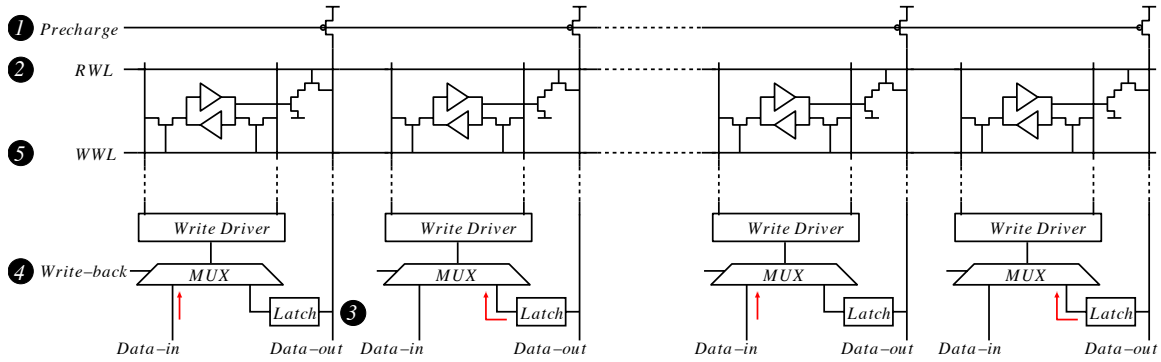

Figure 3.7 - RMW

In summary, RMW operations consist of reading a row, partially modifying the row and finally writing back to the SRAM arrays. The read operations performed before writes degrade performance and increase cache traffic and consequently power dissipation. In addition, the potential performance improvement due to exploiting dual ports is wasted as the read port is occupied for write requests [37].

Park et al. [37] exploited the hierarchical structure of RBLs to perform RMW locally. They isolated the sub-array performing the write operation from RBLs to service another request simultaneously. However, in this technique the sub-array performing the write operation is not available to any other cache access.

Kim et al. [42] proposed an adaptive WWL pulse width and voltage modulation technique to address dynamic write failure. They modulated WWL pulse width and voltage level to ensure that all cells are written successfully.

In this thesis, we focus on RMW and show that a significant improvement in power dissipation and performance can be achieved by considering application behavior and the stream of cache requests. We present our motivations in the following section.

3.3 Motivation

In Figure 3.8 we show how RMW impacts L1 data cache's (implemented using 8T cells) access frequency and performance compared to the conventional 6T cells caches (detail in Section 3.5). In this figure, the left bar shows cache access frequency increase (on the left axis) and the right bar shows performance slowdown (on the right axis). As presented, on average cache access frequency increases by more than 31% while IPC drops by 15%. There is a correlation between cache access frequency increase and performance slowdown almost in all benchmarks, except in *omnetpp* and *soplex*. We will show later that for these two benchmarks, performance is not sensitive to cache access latency. Hence, increasing cache access frequency does not have a significant impact on performance.

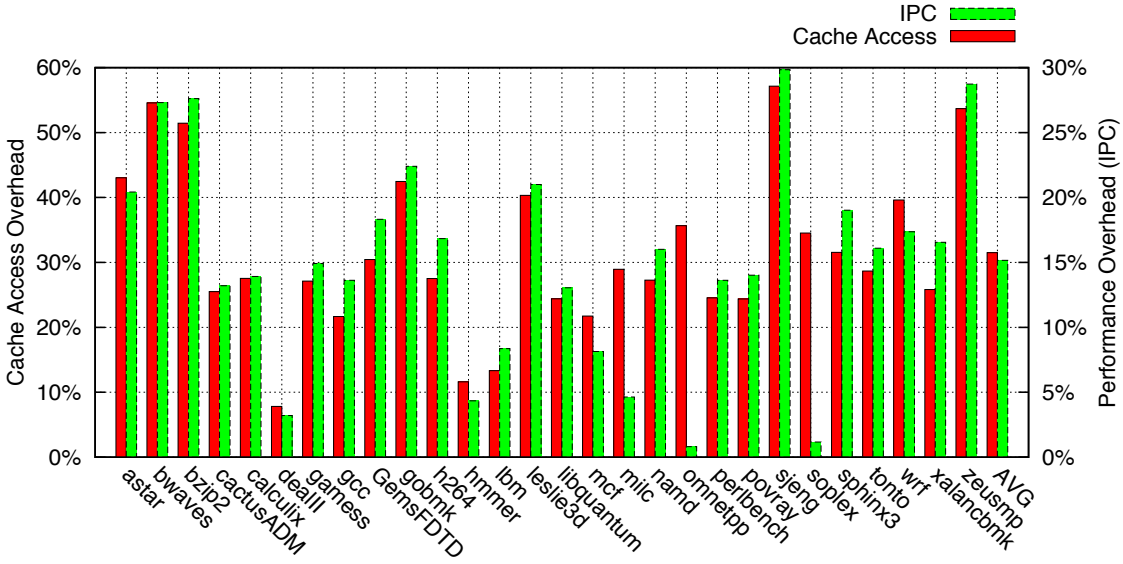

Figure 3.8 - RMW's cache traffic and performance overhead

We identify two improvement opportunities, which can effectively reduce cache access frequency overhead and consequently the associated performance slowdown in RMW. To present the first improvement opportunity, in Figure 3.9 (left side) we show the sequence of operations needed by RMW for performing three consecutive write requests to the same cache sets. As depicted, for each write, three operations (read, modify, and writeback) are required.

We refer to the stream of consecutive write requests made to the same cache set with no read request to the same cache set in between as Consecutive Write Stream (CWS). As depicted in Figure 3.9, in RMW and for each write request in the CWS the addressed cache set is read, modified, and finally written back to the cache. By looking closely at this figure, we see that after the first writeback operation the same cache set is read to fill latches (shown at the bottom of Figure 3.7) for performing the second write request. This situation occurs again after the second writeback operation. The sequence of writeback and read operations is inefficient as RMW writes and reads the same cache set repeatedly.

To avoid this sequence, the read cache set can be buffered and delayed till the next cache request (instead of performing writeback immediately). If the next cache request is a write request to the same cache set (extending the size of the current CWS), the previous writeback operation and the following read operation can be avoided and only the modification operation can be performed (we show the required operations in this case in Figure 3.9 on the right side). In this order, we perform only two cache accesses per each formed CWS, one read operation (initiated by the first write request in the

stream) and one writeback operation (for the last write request in the stream). Thus, the larger the formed CWS, the higher the cache access reduction.

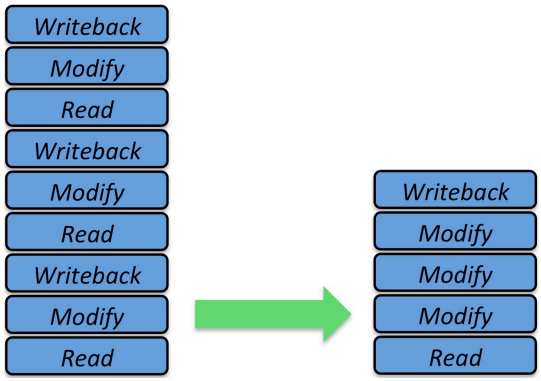

Figure 3.9 - Required operation for a CWS of size 3 under RMW (left) and the required operation after eliminating the redundant operations (right)

To quantify how often cache requests are made to the same cache set we introduce set locality. Set locality shows how frequently the same cache set is accessed by two consecutive cache requests. Our study shows that on average more than 15% (max: 25%) of cache accesses are made to the same cache set. We further breakdown same-set cache accesses into four possible combinations. These combinations include Read-Read (RR), Write-Write (WW), Read-Write (RW), and Write-Read (WR). RR refers to the scenario where both consecutive accesses to the same cache set are reads. Similarly in WW, both cache accesses are writes. Two other scenarios are RW and WR, which refer to read then write and write then read, respectively. In Figure 3.10, we show how often these scenarios occur in SPEC 2006 benchmarks [43]. Our simulation results also show that set locality is relatively insensitive to cache size and configuration.

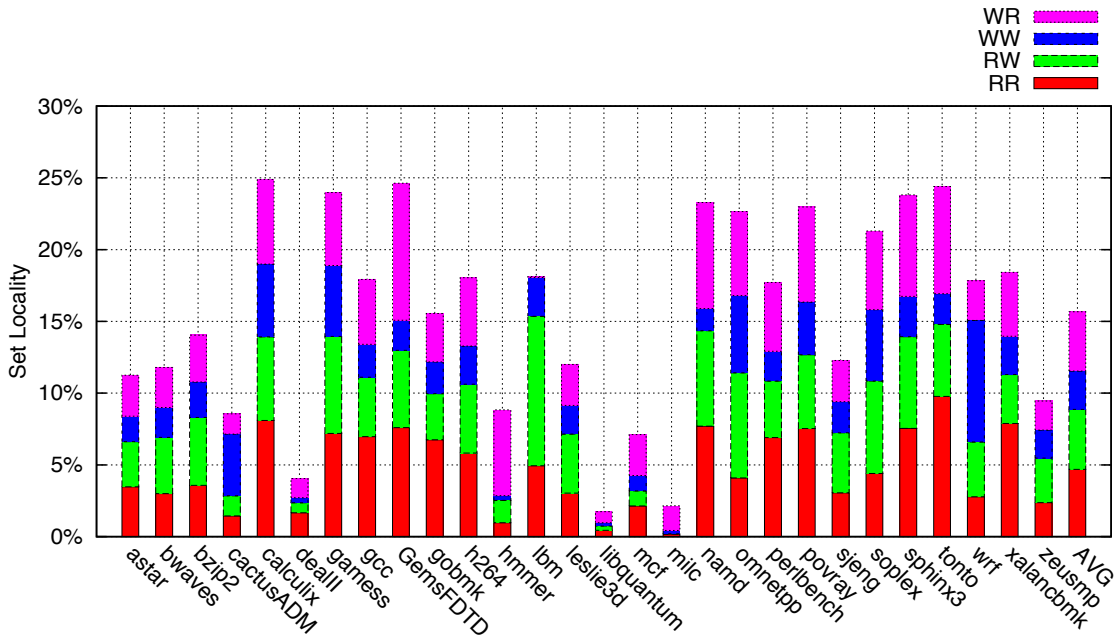

Figure 3.10 - Set locality in SPEC2006 benchmarks

To provide better understanding, we also report set locality for different parallel processing benchmarks. We show set locality in eight benchmarks from PARSEC suite, a modern parallel processing benchmarks, in Figure 3.11. As this figure shows, on average set locality is more than 17%, which is slightly higher than SPEC2006 benchmarks. This figure also shows that set locality could also be found in other application types.

In addition to the first improvement, the second improvement in RMW can be achieved by detecting silent writes [44] and avoiding the associated writeback operations [41]. Silent write refers to a write request that writes the value that is already stored, hence not making any difference. In other words, silent writes do not change the memory content. Therefore, the execution of silent writes can be avoided without compromising coherency and correct execution concerns [41, 45]. Silent writes can be simply detected

in the modification step in RMW by comparing the old value with the new value. In case of a match (silent write), the following writeback operation can be avoided. In Figure 3.12, we show the frequency of silent writes in SEPC 2006 benchmarks. As presented, on average more than 48% (max: 96%) of write requests are silent.

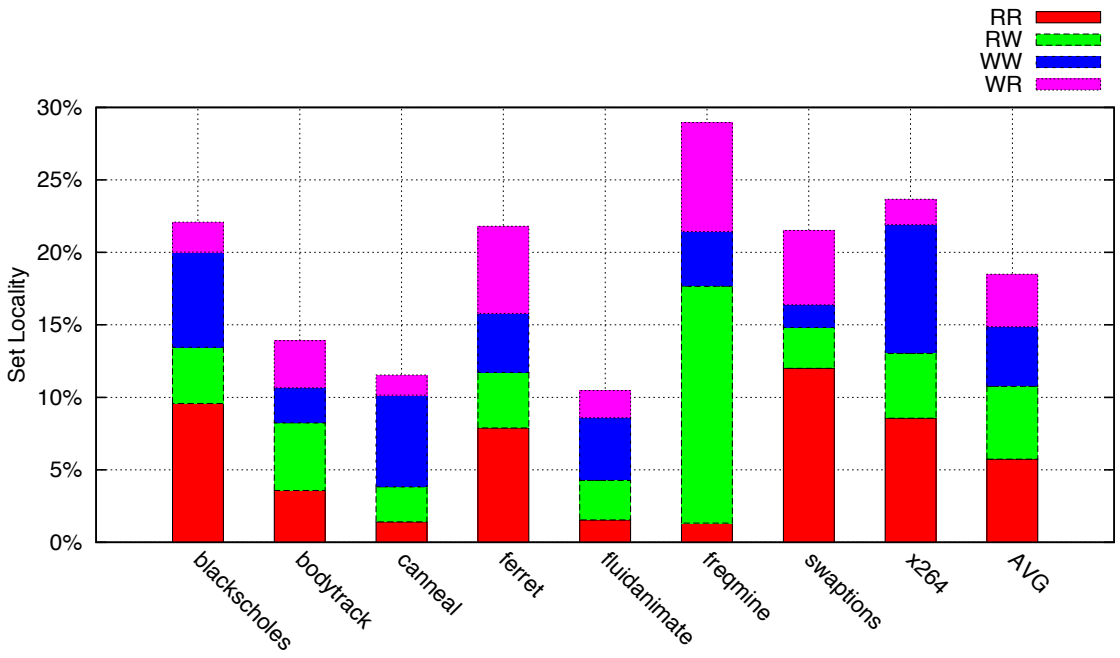

Figure 3.11 - Set locality in PARSEC applications

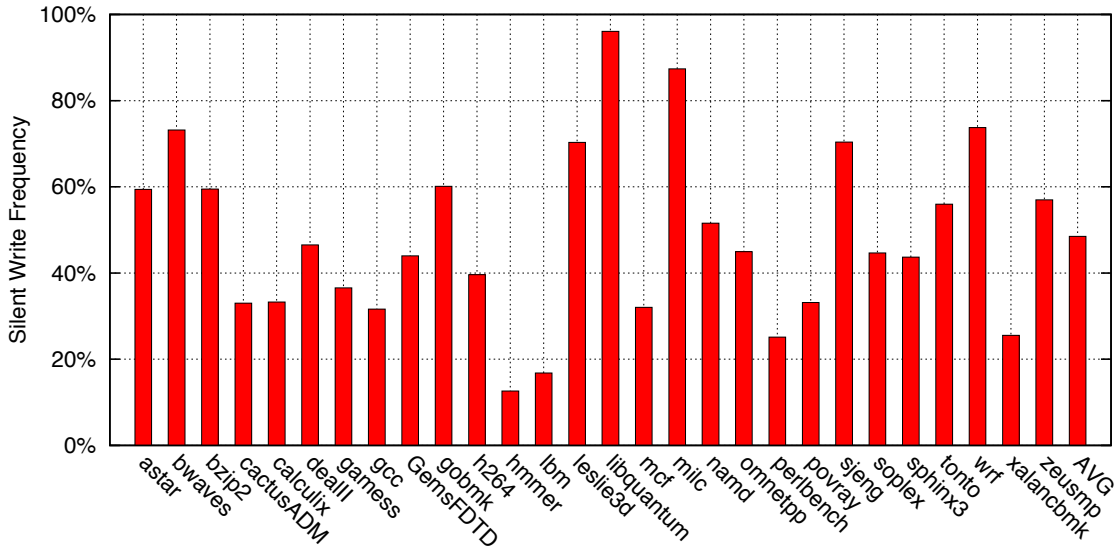

Figure 3.12 - Silent write frequency

3.4 Solutions

In this section, we introduce our proposed techniques. Our first solution, WG, takes advantage of the two aforementioned improvement opportunities by aiming at forming CWS and performing only one read operation for the formed CWS. In addition, WG keeps track of silent writes in the formed CWS and avoids the writeback operation, provided all writes in the formed CWS are silent. We also propose WG+RB, which enhances WG's effectiveness further at a negligible area cost. We explain our proposed techniques in the following sections in more detail.

3.4.1 Write Grouping (WG)

WG exploits two buffers, the Set-Buffer and the Tag-Buffer, to form CWSs. As depicted in Figure 3.13, the Set-Buffer is added to the SRAM array between the multiplexers and the write drivers with no impact on the read path. Unlike RMW, data coming from the latches is stored in the Set-Buffer and the following writeback

operations are delayed till the next cache request without blocking the read path. Thus, the cache is available for servicing the next request while the writeback operation is not performed. To keep track of addresses, we store the cache set number and all cache block tag addresses in the Tag-Buffer located in the cache controller as depicted in Figure 3.14.

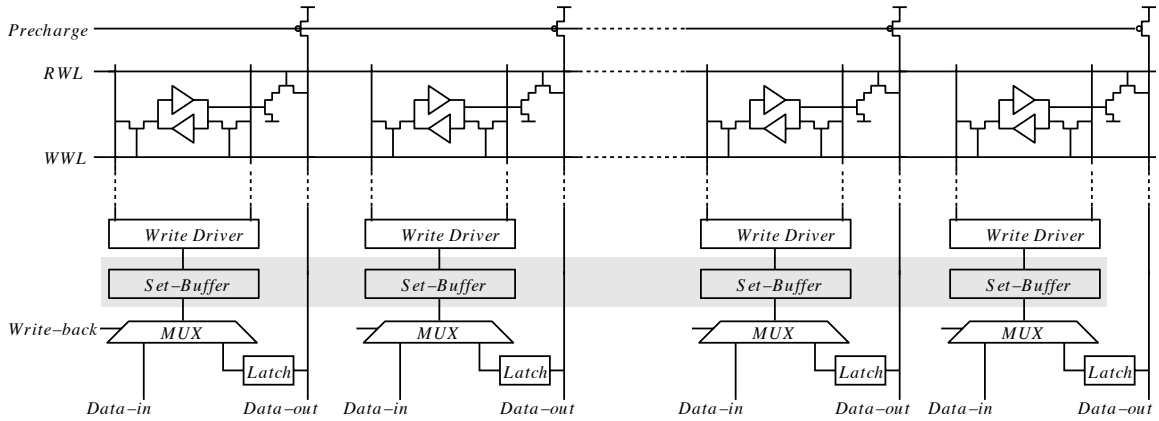

Figure 3.13 - WG block diagram

We also augment the cache controller with a single bit, the Dirty bit (shown in Figure 3.14). While the Set-Buffer forms CWSs, information regarding silent writes in the formed CWSs is maintained by the Dirty bit. The initial value of the Dirty bit is zero and the cache controller sets the Dirty bit to one whenever the write request is not silent. To detect silent writes, during the modification operation the cache controller compares the current value of the Set-Buffer with the value that is being written. The Dirty bit is set if the comparison results in a mismatch. Hence, a zero Dirty bit indicates that all write requests in the formed CWS are silent.

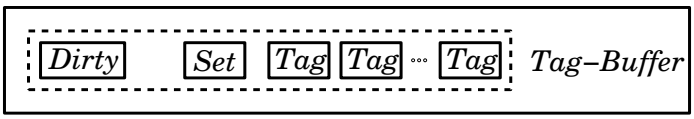

Figure 3.14 - Tag-Buffer and the Dirty bit

To perform a write request, similar to RMW, the addressed cache set is read first. Then, the Set-Buffer is filled with the latches' data and the cache controller modifies the Set-Buffer in the selected columns. Unlike RMW, the writeback operation is not performed immediately and is delayed till the next cache request. Upon receiving the next cache request, the cache controller probes the Tag-Buffer first. Depending on the request's type (read or write) and request's address four different scenarios are possible. We depict these scenarios in Figure 3.15. The green rectangles located on top of the figure show the received cache requests by the data cache controller and the lower blue rectangles show the operations required under WG. We assume that the first write request has already been made to set A (the lower green rectangles). Therefore, the Tag-Buffer is already filled with the set A's addresses and the Set-Buffer is filled with the cache set A's data. The Dirty bit has been set if the first write request is not silent. In the following, we list the steps required by WG to perform the second cache request:

- *Write to the same cache set:* The second write request hits in the Tag-Buffer. Accordingly, the cache controller updates (modify) the Set-Buffer (no cache access is needed). It is worth to mention that the Dirty bit is set if the second write request is not silent.

- *Read from a different cache set:* The read cache request misses in the Tag-Buffer. Since the read port is available, the read cache access is made to retrieve data from the cache.
- *Write to a different cache set:* The write request misses in the Tag-Buffer. The cache controller evicts a set from the Set-buffer by performing a writeback operation only if the Dirty bit is one. If the Dirty bit is zero, it means that the Set-Buffer has not been changed and the associated writeback operation can be avoided. Then (regardless of performing the writeback operation or not), the new accessed cache set is read and stored in the Set-Buffer and the Tag-Buffer is updated accordingly. The Dirty bit is cleared initially to indicate that the Set-Buffer and the cache are consistent. Finally, the Set-Buffer is updated (modified) according to the second write request. The Dirty bit is set if the second write request is not silent.
- *Read from the same cache set:* The read request hits in the Tag-Buffer. In this scenario, the most recent written value is in the Set-Buffer provided the Dirty bit is one. In order to maintain coherency and guaranty correct execution, WG needs to writeback the Set-Buffer (only if the Dirty bit is one) to update the cache. Then, the Dirty bit is cleared to indicate that the Set-Buffer and the cache are consistent. Finally, the read request proceeds and reads data from the cache.

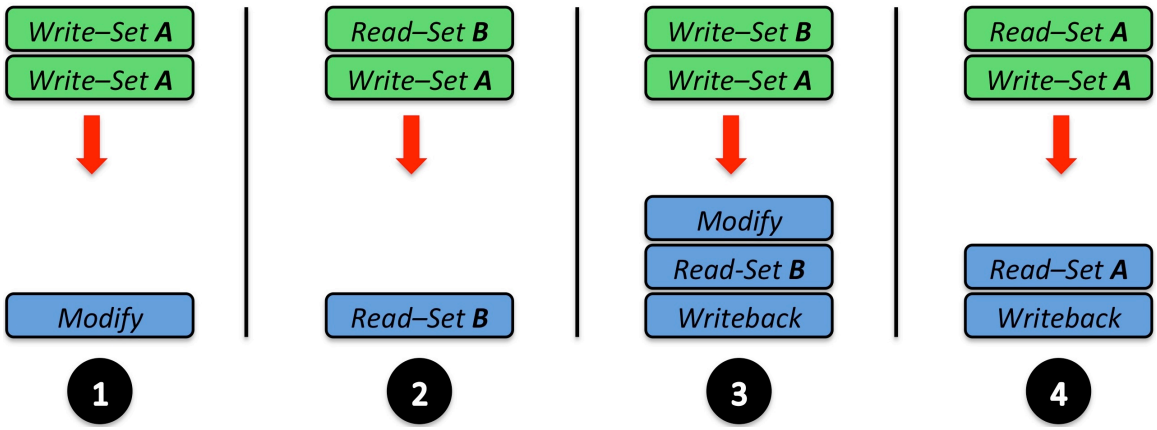

Figure 3.15 - Four possible scenarios in servicing a cache request in WG

In addition to reducing cache access frequency, WG has an impact on cache access latency and consequently performance. Probing the Tag-Buffer for each cache request imposes extra latency on the overall cache access latency that can harm performance. On the other hand, the eliminated fraction of cache accesses can reduce the overall cache access latency and improve performance. We show the simulation results in Section 3.5.

3.4.2 Write Grouping and Read Bypassing (WG+RB)

By comparing the required operations by WG and RMW, we see that the writeback operation is eliminated in the first and second scenarios completely (depicted in Figure 3.15). In the third and fourth scenarios, the writeback operation can be eliminated as well if the Dirty bit is zero. In the third scenario, the writeback operation is inevitable (if the Dirty bit is one) as the write path is blocked. However, in the fourth scenario, the writeback operation is needed only to update the cache. We relax the definition of CWS and propose WG+RB. In the new definition, CWS refers to the stream of consecutive write requests made to the same cache set (no restriction on any read operations inside

the stream). Figure 3.16 shows WG+RB's block diagram. We add a multiplexer to each column of the SRAM array. The *Bypass* signal (controlled by the cache controller) decides to drive the *Data-out* either by the Set-Buffer or the RBL. WG+RB is similar to WG except for the fourth scenario shown in Figure 3.15. In this case, the cache controller configures the multiplexers (by controlling the *Bypass* signal) to pass the Set-Buffer's data to the Data-Out. Therefore, the most recent written value is retrieved from the Set-Buffer. It is worth to mention that in this case two cache accesses are eliminated (one writeback and one read).

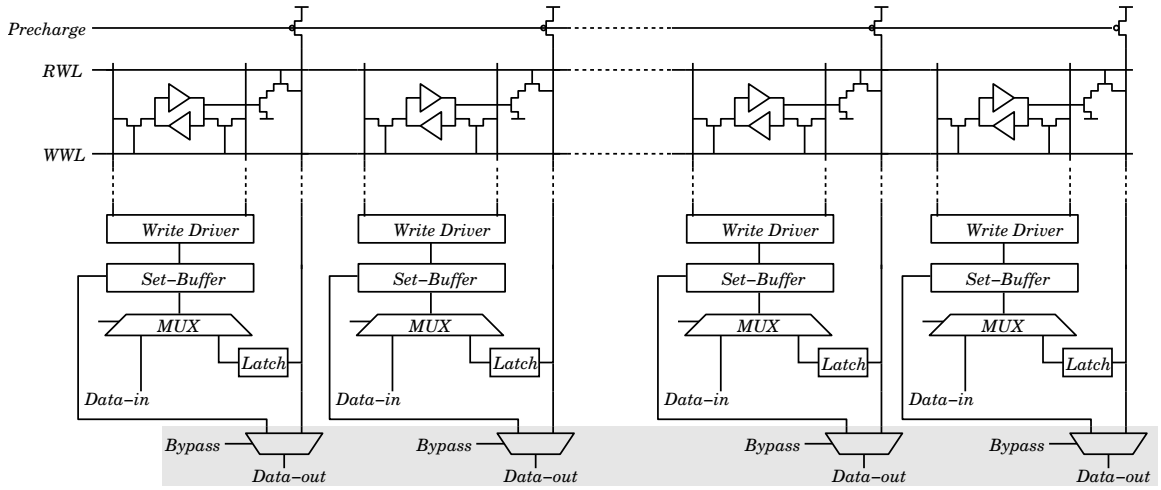

Figure 3.16 - WG+RB block diagram

In addition to the extra cache access frequency reduction by WG+RB, performance is expected to improve further compared to WG for two reasons. First, the preceding writeback operation is eliminated. Then, bypassing data through the multiplexers is faster than reading data from the cache. We report WG+RB's impact on cache traffic and performance in Section 3.5.

3.4.3 Example

To provide better understanding, we present an example in Figure 3.17. To simplify the example, we consider only two cache sets, set *a* and set *b*. In order to differentiate between read and write requests we use *R* and *W*. For example, R_a and W_b indicate reading from set *a* and writing to set *b*, respectively. In this figure, the first row shows the original stream of cache requests issued by a processor. Requests arrive at the L1 data cache from right to left (the cache receives a read request from set *a*, then a write request to set *b*, and so on). In this example, we also assume that the last write request (*Wa* in the first row) is a silent write.

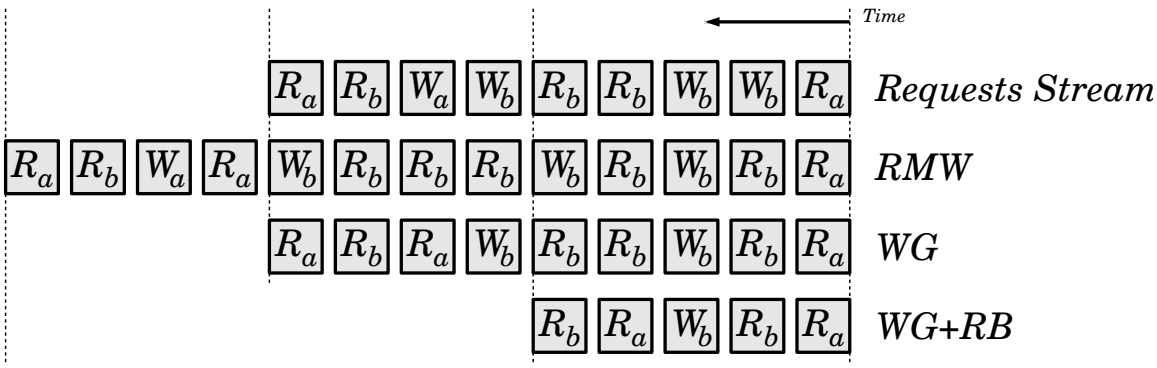

Figure 3.17 - Example

In this figure, the second row shows the required cache accesses under RMW, where each write request is preceded by a read request. In the third row, we show how WG forms CWSs and eliminates unnecessary cache accesses. Initially, the Tag-Buffer is empty. Hence, the first read request R_a misses in the Tag-Buffer and a cache access is performed. The next cache access is W_b and therefore the cache controller issues a read operation (the first R_b in the third row). Then, the Set-Buffer in the selected columns is

filled with data from Data-in and the Set-Buffer in the half selected columns is filled with data read from the latches. In meanwhile, WG compares the previous value with the new value to detect silent writes. In this case, the Dirty bit is set as the current write request is not silent. Unlike RMW, WG does not perform the writeback operation immediately and waits till the next cache request. The next request (the second W_b in the first row), hits in the Tag-Buffer and only the Set-Buffer in the selected columns is updated. The following requests (the first and the second R_b in the first row) are read from set *b*. Hence, the cache controller issues a writeback operation (the first W_b in the third row) to update the cache and then clears the Dirty bit. Thereafter, the two read operations are performed. The next request (the third W_b in the first row) hits in the Tag-Buffer and updates the Set-Buffer. The cache controller sets the Dirty bit to indicate that the Set-Buffer is modified.

The next request (the first W_a in the first row) misses in the Tag-Buffer and the cache controller updates the cache by issuing another writeback operation (the second W_b in the third row) and a read operation (the second R_a in the third row) to fill the Set-Buffer with set *a*'s data. Since the current write request is silent, the Dirty bit is not set by the cache controller. Upon the next request (the third R_b in the first row), a Tag-Buffer miss occurs and since this is a read operation a cache access is performed. Although the last request (the second R_a in the first row) is a read request and hits in the Tag-Buffer, the cache controller does not initiate the writeback operation to update cache since the Dirty bit is clear (the cache and the Set-Buffer are consistent). Therefore, the last R_a in the third row is performed to retrieve data from the data cache.

We demonstrate how WG+RB deals with the same request stream in the fourth row in Figure 3.17. There are only two differences between the WG and WG+RB. First,

WG+RB avoids the first W_b and two subsequent R_b in WG, as the most recent written data are retrieved from the Set-Buffer through the multiplexers (shown in Figure 3.16) and no writeback operation is needed for updating the cache. Second, the last R_a is avoided by WG+RB as it hits in the Tag-Buffer and is bypassed through the multiplexers.

3.5 Evaluation

3.5.1 Methodology

We use MARSSx86 a full system and cycle accurate multi-core x86 simulator [46] to evaluate our proposed techniques. Table 2 shows the baseline processor configuration. We use SPEC 2006 benchmarks [43] and run all benchmarks for 100M instructions after 1B fast-forwarded instructions. In our proposed techniques, we consider one-cycle latency to probe the Tag-Buffer and one-cycle latency to modify the Set-Buffer. In WG+RB, particularly, we also consider one-cycle latency for passing data from the Set-Buffer to the Data-out.

Table 2 - Processor configuration

Core		***Cache and Memory***	
Fetch/Issue/Commit Width	6/4/4	L1 Data Cache	64KB, 32B Line Size, 4-way, 4-cycle Latency, Write-back
ROB Entries	96	L1 Instruction Cache	64KB, 32B Line Size, 4-way, 4-cycle Latency
Issue Queue Entries	32	Unified L2 Cache	2MB, 64B Line Size, 8-way, 25-cycle Latency, Write-back
Branch Predictor	Tournament, 48KB	Memory	50 ns

Before presenting our simulation results, we briefly study two characteristics of SPEC 2006 benchmarks. The first characteristic that we are interested in is the frequencies of read and write instructions. In Figure 3.18, the lower portion shows the frequency of read

instructions while the upper portion shows the frequency of write instructions. As presented, the average frequencies of read and write instructions are 23% and 8%, respectively. While in most of the benchmarks read instructions are more frequent than write instructions, write frequency of *bwaves*, *bzip2*, *leslie3d*, *sjeng*, *wrf*, and *zeusmp* is very close to their read frequency. As we showed in Figure 3.8, the cache traffic overhead for these benchmarks is high as well.

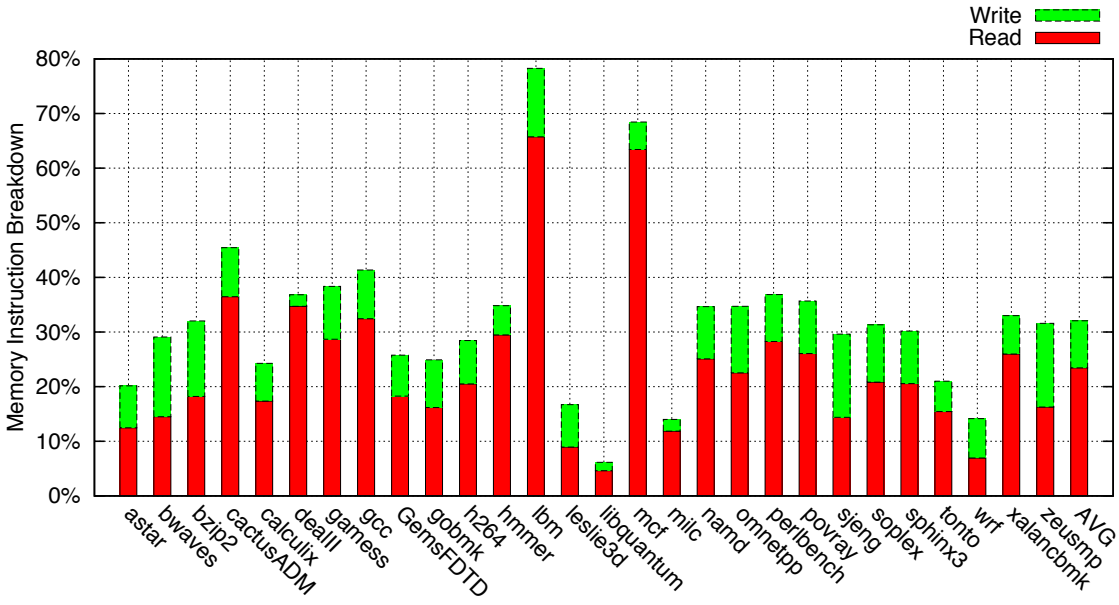

Figure 3.18 - Frequencies of read and write instructions

The second characteristic we are interested in is performance sensitivity to cache access latency under RMW. In Figure 3.19 we compare performance (IPC) improvement for an L1 data cache with one-, two-, and three-cycle access latency normalized to our baseline (four-cycle). As this figure shows, performance improves on average by 20%, 49%, and 90% for access latency of three-, two-, and one-cycle, respectively. For one-cycle cache latency, *namd* and *gamess* experience the highest performance improvement

by 143% and 133%, respectively. On the contrary, *omnetpp* and *soplex* show the lowest performance improvement (below 1%) even for one-cycle cache access latency. This shows that performance in these two benchmarks is not sensitive to the L1 data cache access latency.

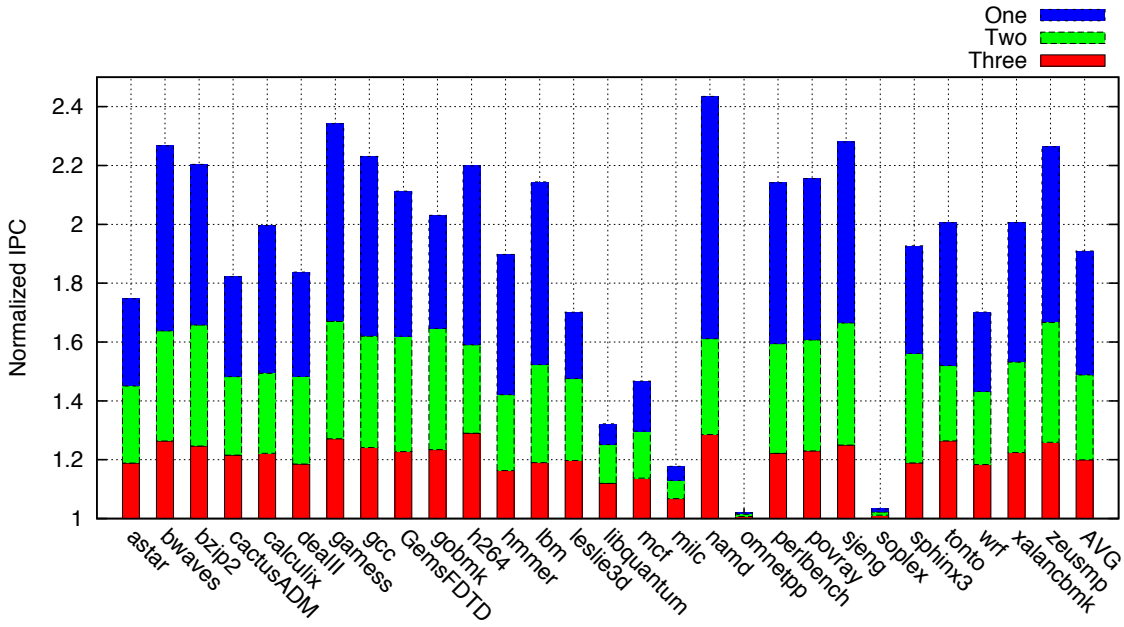

Figure 3.19 - Performance sensitivity to the L1 data cache access latency under RMW

3.5.2 Cache Access Frequency Reduction

We present cache access frequency reduction in Figure 3.20. In this figure, the left bar (labeled as Set-Buffering) shows cache access reduction when silent write detection mechanism is disabled. In other words, Set-Buffering shows cache access reduction when we perform writeback operation only when needed (scenarios 3 and 4 in Figure 3.15) regardless of if all write operations performed on Set-Buffer are silent or not. The middle and right bar show cache access frequency reduction achieved by WG and WG+RB, respectively. On average cache access frequency is reduced by 10%, 15% and 20% by

employing Set-Buffering, WG and WG+RB, respectively. Note that the gap between Set-Buffering and WG shows how detecting silent writes can improve WG.

We achieve a significant cache access frequency reduction (roughly 35%) in *wrf* by employing WG and WG+RB. This could be explained by referring to Figure 3.18, Figure 3.10, and Figure 3.12. First, as we show in Figure 3.18, *wrf*'s read and write frequencies are around 7%. In other words, half of its memory instructions are write requests. Second, as presented in Figure 3.10, the WW share is highest (more than 8%) for *wrf*, which increases the chance of forming larger CWSs. And finally, *wrf* is the third silent write frequent (more than 73%) benchmark. Accordingly, WG and WG+RB are able to form larger CWSs and due to high silent write frequency, a higher number of associated writeback operations can be eliminated. Similar conclusions can be made for other benchmarks, especially for *bzip2* and *sjeng*.

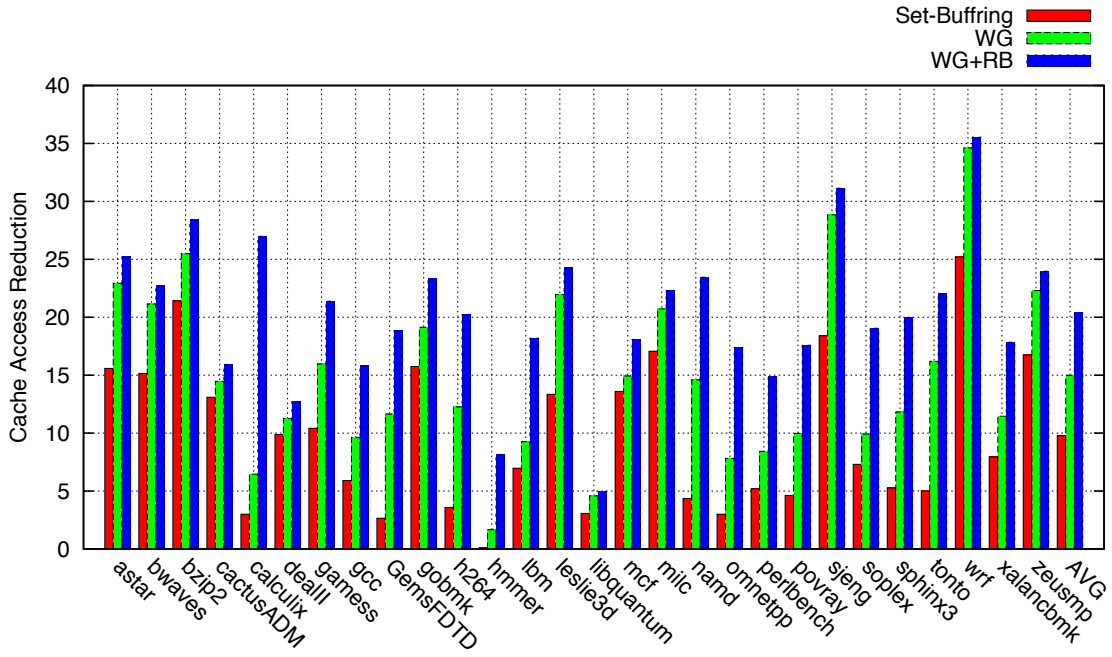

Figure 3.20 – Set-Buffering, WG and WG+RB cache access reduction

While WG+RB outperforms WG in reducing cache access frequency across all benchmarks, its effectiveness is not constant and varies from benchmark to benchmark. For example, *hmmer* benefits significantly by employing WG+RB because the WR (write then read) scenario is more frequent compared to other scenarios (refer to Figure 3.10). Therefore, WG+RB is able to avoid a large number of read requests that come after writes to the same cache set. Note that although *libquantum* has the highest frequency of silent writes, it does not benefit from WG and WG+RB. The first reason is low write frequency (shown in Figure 3.18). The other reason is low set locality (refer to Figure 3.10). Therefore, the chance of forming large size CWSs is low.

3.5.3 Performance

In Figure 3.21, we show performance improvement for Set-Buffering, WG and WG+RB. In this figure, we normalize IPCs to the baseline (RMW) IPC. As this figure shows, on average Set-Buffering, WG and WG+RB improve performance by 23%, 31% and 37%, respectively. As we expected, WG+RB outperforms WG in all benchmarks. The highest performance improvement is achieved in *bwaves* by 75% and 78% by employing WG and WG+RB, respectively. Two benchmarks, *omnetpp* and *solpex*, experience the lowest performance improvement. As we showed earlier in Figure 3.19, this is because of the low sensitivity of these two benchmarks' performance to the cache access latency. Thus, they do not benefit from WG and WG+RB. Note that although WG and WG+RB could not improve performance for these two benchmarks, they can still reduce cache access frequency considerably (see Figure 3.20).

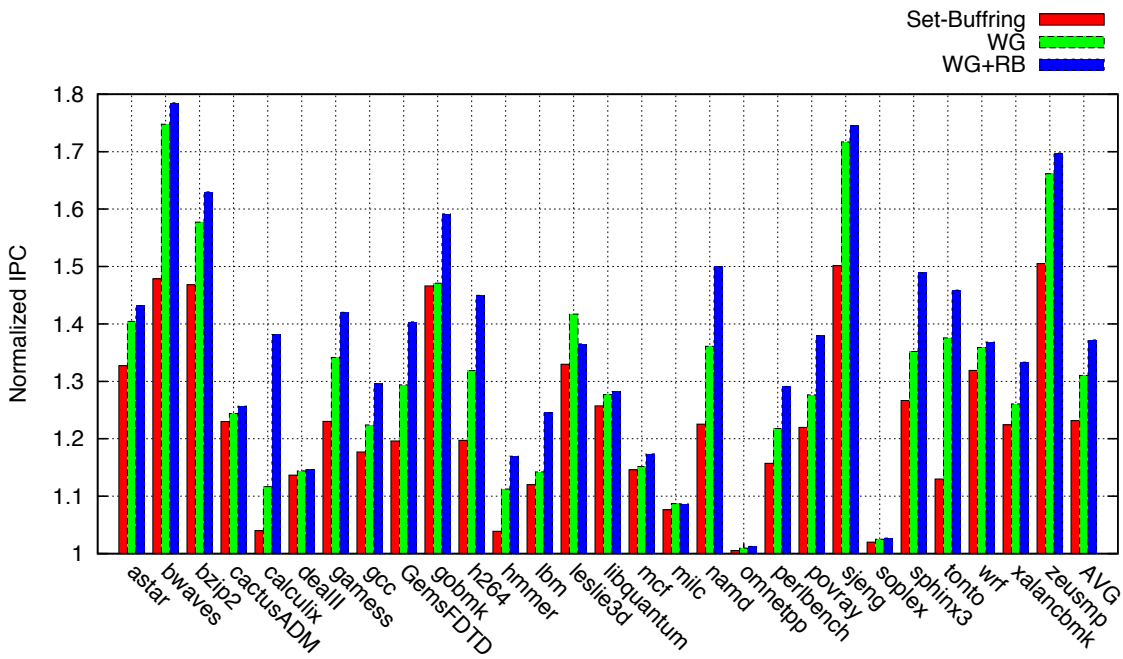

Figure 3.21 – Set-Buffering, WG and WG+RB performance improvement

3.5.4 Area and Power Overhead

WG and WG+RB impose the extra area overhead associated with the Set-Buffer and the Tag-Buffer. The Set-Buffer size is equal to the size of one cache set. For example, in our baseline cache (64KB, 4-way, and 32B block size), the size of a cache set is 128B. Accordingly, the Set-Buffer imposes less than 0.2% area overhead compared to the overall cache size. In this case, the Tag-Buffer size is negligible (less than 150 bits assuming 48 bits physical address). In addition to storage overhead, WG and WG+RB require comparators (equal to the number of columns in each row, here 128) to detect silent writes. WG+RB needs extra multiplexers (128 2:1-multiplexers) to route data from the Set-Buffer to the Data-out.

Furthermore, probing the Tag-Buffer and accessing the Set-Buffer dissipates power and could impact the overall power dissipation. In this work, we assume that power dissipation of the Tag-Buffer and Set-Buffer is negligible compared to the overall cache power dissipation.

3.5.5 Discussion

As we discussed earlier, although employing 8T cells in caches improves cell stability and area efficiency in future technology nodes compared to 6T cells, addressing the column selection issue by employing RMW is very expensive in terms of cache traffic and performance. We also showed that our proposed techniques, WG and WG+RB, reduce the overhead imposed by RMW significantly. In this section, we compare cache traffic and performance of our proposed techniques to the conventional 6T cells caches.

In Figure 3.22, we compare cache traffic of WG and WG+RB to 6T cells caches. As presented, cache access frequency increases on average by 11% and 4% by employing WG and WG+RB, respectively. Note that, in some benchmarks (e.g., *dealII* and *wrf*) WG and WG+RB reduce cache access frequency compared to 6T cells caches.

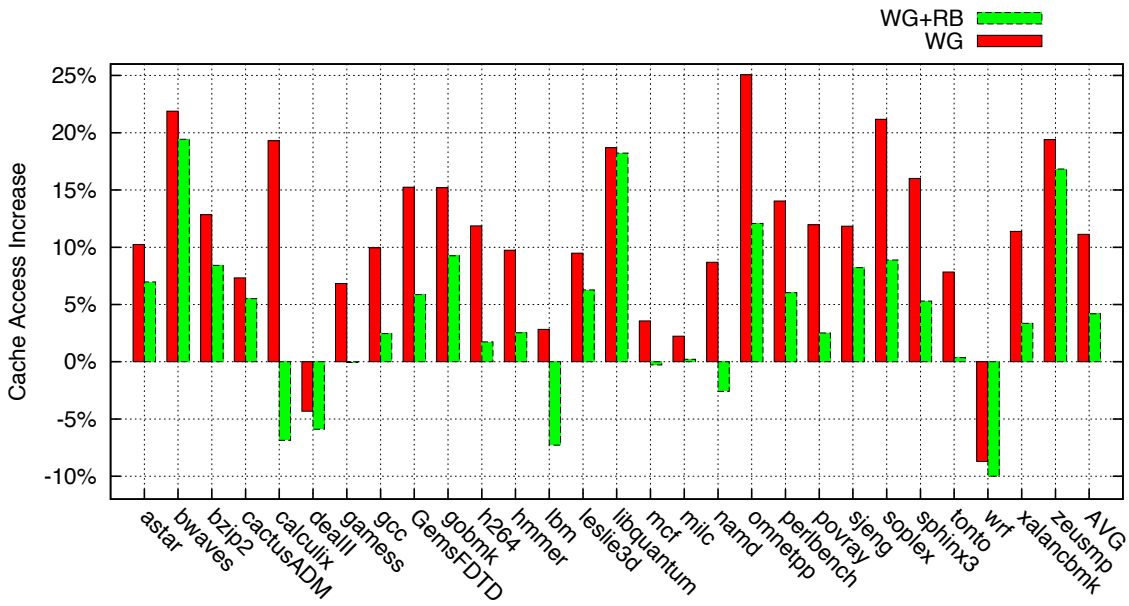

Figure 3.22 - WG and WG+RB cache traffic compared to 6T cells caches

In Figure 3.23, we depict the impact of WG and WG+RB on performance (IPC) normalized to 6T cells caches. As presented, average performance improves by 10% and 15% in WG and WG+RB, respectively. The highest performance improvement is achieved by *bwaves*. Consistent to our previous findings, no significant performance improvement can be achieved for *omnetpp* and *soplex*.

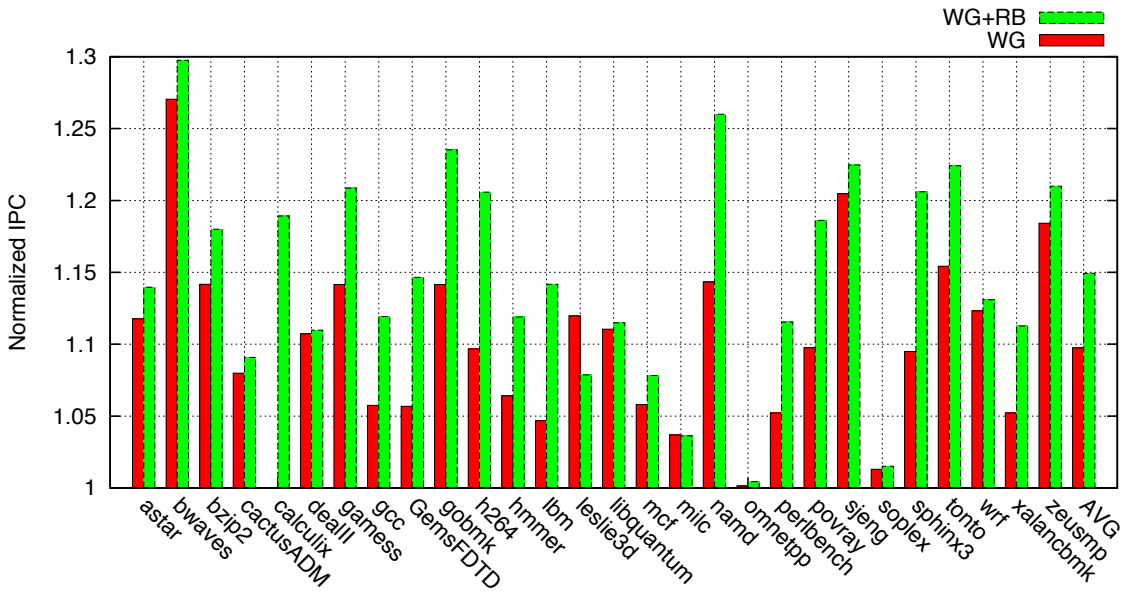

Figure 3.23 - Performance improvement achieved by WG and WG+RB compared to 6T cells caches

To summarize, our simulation results show that employing WG and WG+RB in 8T cells caches has two advantages over 6T cells caches. First, a higher level of cell stability can be achieved by using 8T cells. Second, WG and WG+RB improve performance compared to 6T cells caches.

3.6 Conclusion

Conventional 6T cells caches can suffer from cell instability. Cell stability limits the potential power saving achievable by voltage scaling. While proposed 8T cells enhance stability, they suffer from column selection issue during write operations. RMW addresses this issue at the cost of an increase in the number of cache accesses (one extra read before each write) resulting in higher power dissipation and performance loss. In this

chapter, we showed that RMW impacts cache access frequency (increases by 31% on average) and performance (reduces by 15% on average) negatively.

We showed that a considerable share of cache accesses in RMW is unnecessary. To identify and address this inefficiency, we proposed two solutions, WG and WG+RB. By exploiting a buffering mechanism, WG forms the stream of consecutive write requests to the same cache set and performs only one read (instead of one read for each write) for the formed streams. In addition, WG keeps track of silent writes in the formed streams and avoids the write operations if all write requests in the streams are silent. We further enhance the effectiveness of WG by proposing WG+RB, which eliminates a subset of read operations at the expense of a negligible area overhead. Our simulation results show that WG and WG+RB reduce cache access frequency by 15% and 20%, respectively. In addition to cache traffic improvement, WG and WG+RB improve performance by 31% and 37%, respectively.

Chapter 4

Leakage Power Reduction in L1 Data Caches Implemented using 8T Cells

4.1 Introduction

As we discussed earlier in Chapter 2, cache leakage power contributes to overall chip power dissipation significantly. This share is expected to grow by each technology advancement due exploiting lower threshold transistors [47]. We also reviewed cache leakage power reduction techniques and explained that dual threshold techniques are commonly used to reduce leakage power at the cost of performance penalty. The proposed solutions focused on 6T cells caches in which dual threshold technique can impact both read and write latencies.

In Chapter 3, we showed that 6T cells can suffer from cell instability and may not be the best choice for future technologies. Then, we studied 8T cells, which is a promising alternative for implementing caches. Although 8T cells enhance cell stability and robustness to process variations, they suffer from column selection issue during write operations. Accordingly, RMW was proposed to address this issue, however, at the expense of cache traffic and performance penalty. We also identified RMW inefficiencies and proposed WG and WG+RB to reduce RMW's overhead. WG and WG+RB eliminate unnecessary and redundant cache accesses by exploiting a buffering mechanism.

Despite the improvement in cell stability, leakage power is still high in 8T cells for the same reason as 6T cells. In this chapter, we focus on reducing leakage power in L1 data caches employing 8T cells by exploiting dual threshold technique. We are motivated by

the low sensitivity of performance to write latency of L1 data cache. We show that performance degrades significantly by increasing read latency while a modest increase in write latency does not harm performance. We further study performance sensitivity to write latency under our proposed techniques, WG and WG+RB, in 8T cells caches. We show that WG and WG+RB reduce performance sensitivity to write latency effectively.

Based on our observations, we propose a new dual threshold 8T cell that improves leakage power significantly. Exploiting high threshold transistors allows reducing leakage current at the cost of performance overhead. Our proposed 8T cell uses high threshold transistors only in the write path with no impact on the read path. Therefore, an increase in write latency is expected while read latency remains intact. Due to low sensitivity of performance to write latency, we show that the imposed performance overhead is affordable. In addition to leakage power reduction, we show that our proposed cell improves cell stability and is more robust under process variations.

The rest of this chapter is organized as follows. We present our motivations in Section 4.2. We introduce our proposed cell in Section 4.3. In Section 4.4, we present our simulation results. Finally, in Section 4.5, we offer our concluding remarks.

4.2 Motivation

We are motivated by the low sensitivity of performance to slight increases in write latency in L1 data cache. To present our motivation, we study SPEC 2006 benchmarks and selectively increase the read or write latencies of L1 data cache and show their impact on performance. In Figure 4.1, we show performance (IPC) reduction as read and write latencies increase by one and two cycles. As depicted, on average IPC drops by 8% and 15% when read latency increases by one cycle and two cycles, respectively. The

average performance slowdown for increasing write latency by one cycle and two cycles is 4% and 8%, respectively.

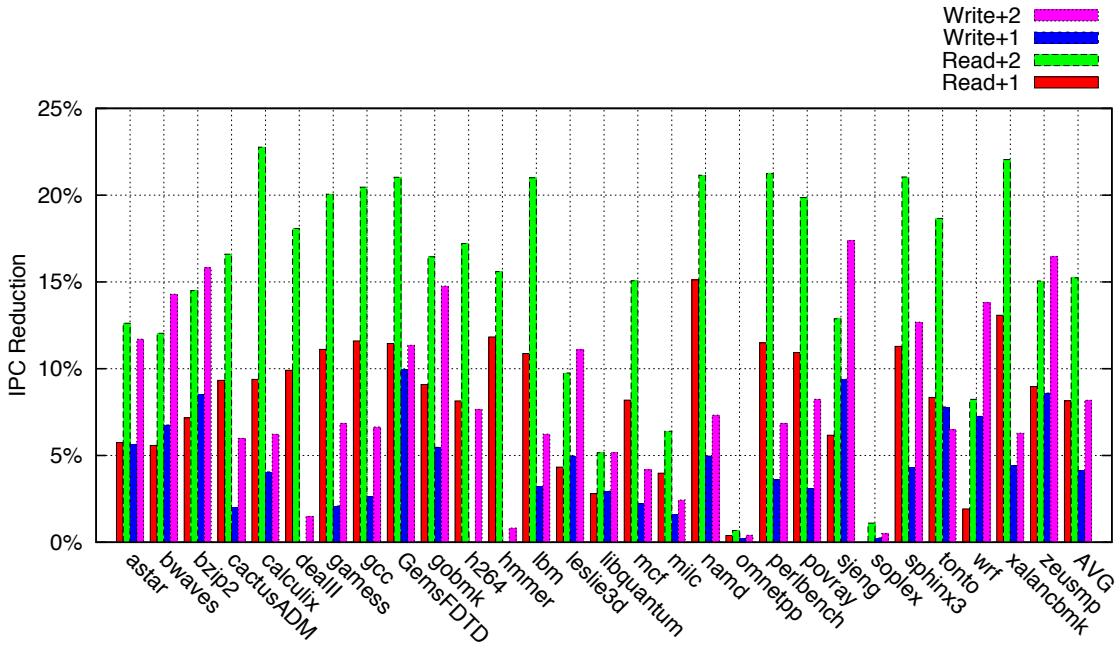

Figure 4.1 - Performance sensitivity to the read and write latencies

In most benchmarks, performance is more sensitive to read latency compared to write latency. For example in *calculix*, IPC is lost by more that 22% when read latency increases by two cycles. At the same time increasing write latency by two cycles reduces IPC by 6%. This figure shows that a slight increase in write latency is more tolerable by processors compared to read latency. This can be explained by the fact that read instructions provide data for their dependent instructions and consequently remove their dependencies. Hence, delaying the read instruction execution reduces instruction level parallelism and results in performance slowdown. However, with the exception of read

instructions, there is no other producer/consumer dependency relation between writes and other instructions. Accordingly, a modest increase in write latency does not harm performance dramatically. The read/write dependency is also addressed often by the forwarding mechanism used in the LSQ or L1 data caches [1].

On the contrary, in a few benchmarks (e.g., *bwaves*, *bzip2*, and *sjeng*) we observe a higher level of performance sensitivity to write latency compared to read latency. Our simulation results show (see Figure 3.18) that these benchmarks are write intensive, and accordingly increasing write latency occupies the cache port for a longer time and consequently reduces cache port availability. By reducing cache port availability, subsequent read instructions are stalled which adversely impacts performance.

Considering this observation, we further investigate performance sensitivity to write latency for 8T cells caches under RMW as shown in Figure 4.2. In this figure, the first and the second bar show performance slowdown when write latency increases by one cycle and two cycles, respectively. This figure shows that average performance loss is about 4% and 8% when write latency increases by one and two cycles, respectively. As we expected, performance overhead is higher in write intensive benchmarks.

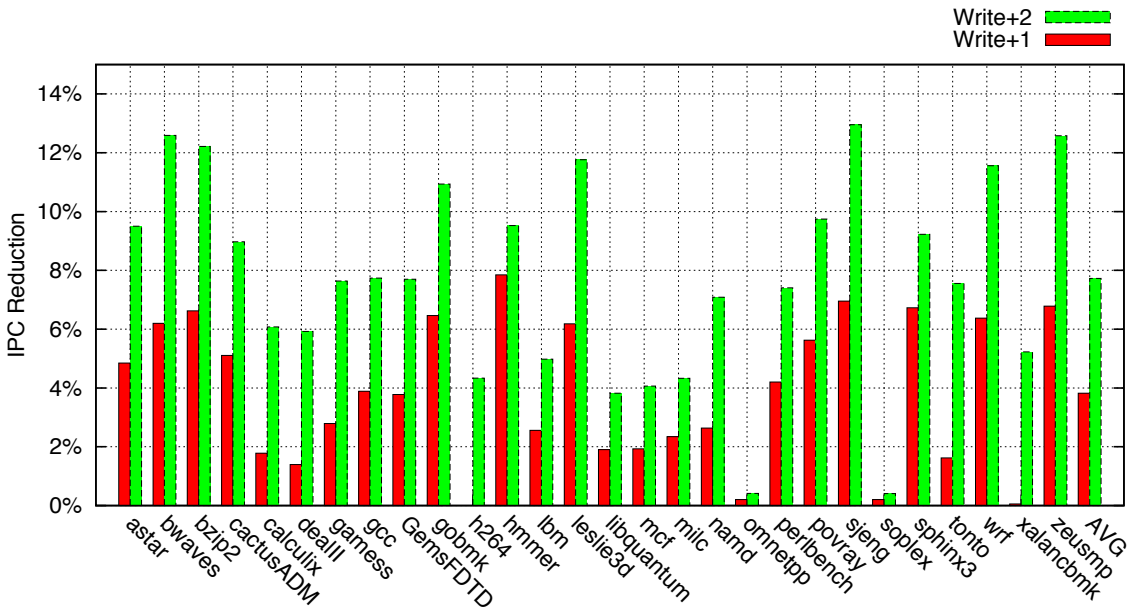

Figure 4.2 - RMW's performance sensitivity to write latency

In addition to RMW, we perform a similar study for 8T cells caches under our proposed techniques, WG and WG+RB. As we stated in the Chapter 2, WG and WG+RB form groups of consecutive write requests and eliminate a considerable amount of cache accesses. Therefore, we expect that performance sensitivity to write latency be lower in WG and WG+RB compared to RMW. In Figure 4.3, we show performance reduction when write latency increases by one and two cycles under WG. As depicted, average performance loss is less than 2% and 4% when write latency increases by one and two cycles, respectively.

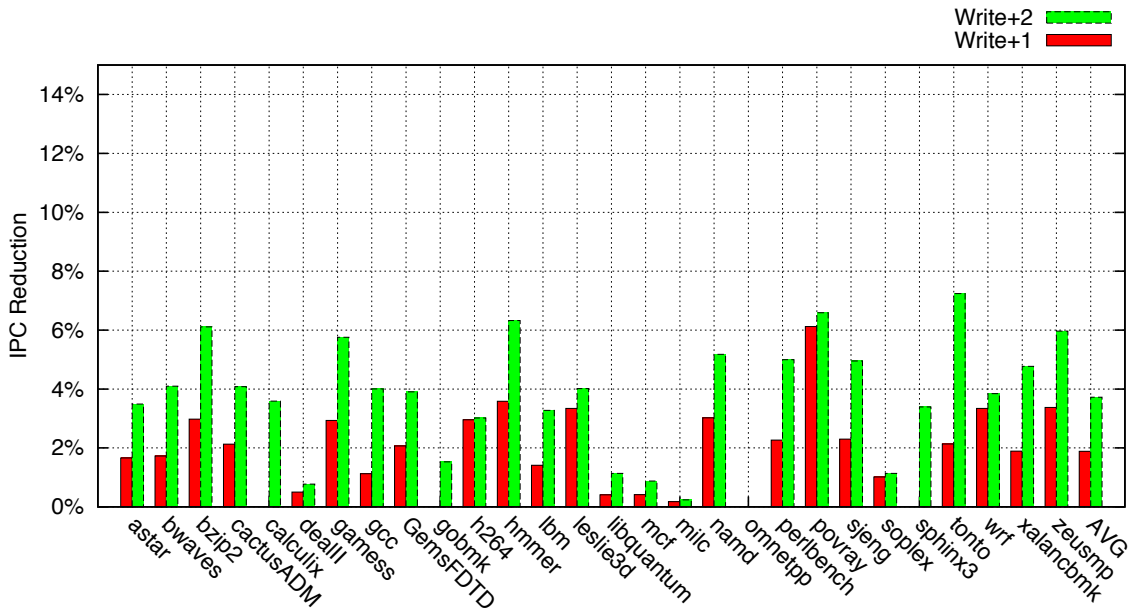

Figure 4.3 - WG's performance sensitivity to write latency

Similarly, we show performance reduction under WG+RB in Figure 4.4. As this figure shows, average performance loss is 2% and 4% when write latency increases by one cycle and two cycles, respectively.

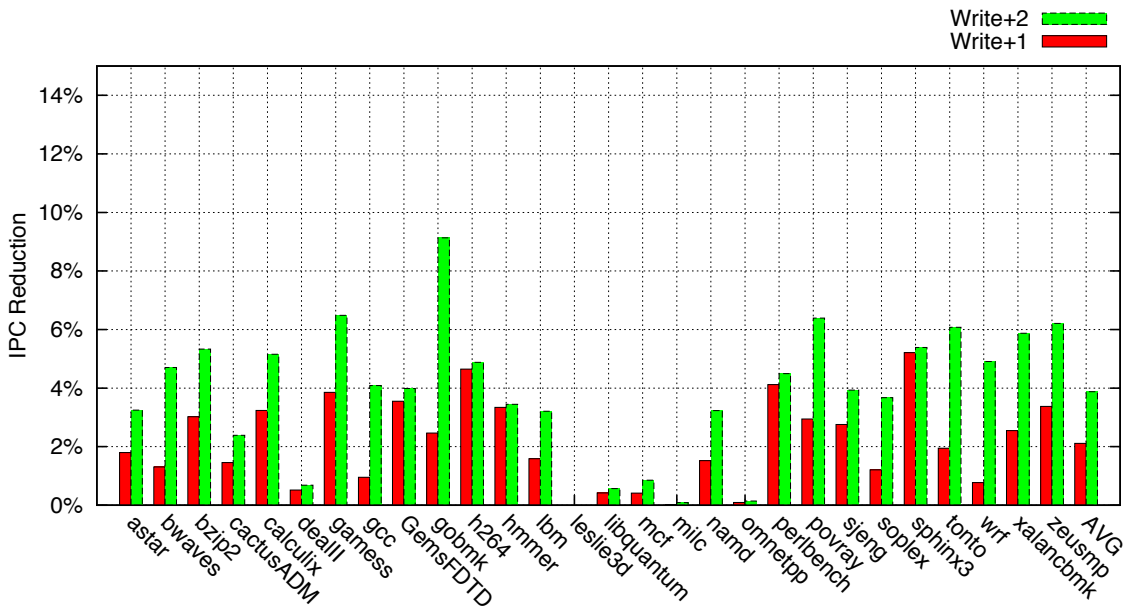

Figure 4.4 - WG+RB's performance sensitivity to write latency

To summarize our findings, we showed that performance is less sensitive to write latency compared to read latency as presented in Figure 4.1. We also showed (in Figure 4.2, Figure 4.3, and Figure 4.4) that WG and WG+RB reduce the negative impact of an increase in write latency on performance. In other words, performance in 8T cells caches under WG and WG+RB is less sensitive to write latency compared to RMW. These observations motivate us to propose a new dual threshold 8T cell introduced in the following section.

4.3 Dual Threshold 8T Cells

In this section, we present our proposed dual threshold 8T cell by replacing six transistors, M1 through M6 with high threshold voltage transistors as shown in Figure 4.5. Note that M7 and M8 are regular threshold transistors. In this new cell, we only use

high threshold transistors in the write path with no impact on the read path. Therefore, we expect an increase in write latency while read latency remains intact. Henceforward, we refer to the 8T cell which all transistors have regular threshold voltage as a regular 8T cell. Read and write operations in the proposed cell are identical to the regular 8T cell. Accordingly, RMW, WG, and WG+RB can be used to address column selection issue. Note that our proposed cell does not impose any area overhead.

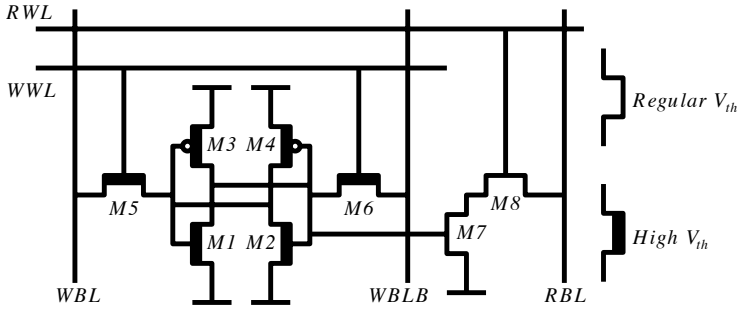

Figure 4.5 - A dual threshold 8T cell

To evaluate the effect of an increase in write latency on performance, we assume that exploiting high threshold voltage transistors in the write circuit increases write latency by one cycle (in an optimistic scenario) and two cycles (in a pessimistic scenario). By referring to Figure 4.2, Figure 4.3, and Figure 4.4, we observe that average performance loss is 4%, 2% and 2% for RMW, WG and WG+RB, respectively under the optimistic scenario. However, under the pessimistic scenario average performance penalty is less than 8% and 4% and 4% for RWM, WG, WG+RB, respectively. It is worth to mention that we show average performance slowdown (due to the extra write latency) for each technique (RMW, WG, and WG+RB) compared to a regular cache under the same technique. For example, in Figure 4.4 we show that WG+RB's performance drops by

around 2% and 4% for one and two cycles extra write latency, respectively. However, in Figure 3.21 we show that WG+RB improves performance by 37% compared to RMW. Therefore, even by using our dual threshold 8T cell under WG+RB, average performance improves by around 36% (for one extra cycle write latency) and 35% (for two extra cycles write latency) compared to RMW. Similar conclusions can be made for WG.

4.4 Evaluation

4.4.1 Simulation Setup

In this study, we use the same methodology presented in Section 3.5.1. For circuit level simulation, we use HSpice and TSMC 90nm library. For modeling process variation, we use Monte Carlo simulation technique with 100,000 samples.

We present the impact of our dual threshold 8T cell on write ability and SNM. There are many techniques to measure write ability [49]. In this work, we use write trip point technique [50] that indicates cell resistance against changing its state during a write operation. Lower write trip point shows the higher cell resistance to change its state. Hence, higher write trip point is preferred. To measure cell stability, we also use SNM (Static Noise Margin) proposed by [52]. SNM refers to the minimum required DC voltage to flip cell state. In other words, SNM shows cell stability in the presence of static noise. Another stability metric is read stability that indicates cell stability during a read operation. Read stability is a very important metric in 6T cells. However, since read and write paths are separated in 8T cells and our proposed cell does not have any impact on the read path, we assume read stability does not change. Therefore, we do not study read stability in this paper.

4.4.2 Results

Our HSpice simulation results show 37% reduction in leakage power by employing our proposed dual threshold 8T cell. We also observe that our proposed cell improves SNM by 10%. This is due to the fact that by increasing threshold voltage a higher noise level is required to switch cell state.

However, on the negative side, our simulation results show 18.8% increase in write latency. We discussed earlier the impact of increase in write latency on performance. Our simulation results also confirm that our proposed cell has no impact on read latency. In addition to write latency, our proposed cell degrades write trip point by 13.5%.

Besides the nominal condition, we study our proposed cell under process variations by using Monte Carlo simulation. In Figure 4.6 we present SNM distribution for our proposed cell (the right bell) and the regular threshold 8T cell (the left bell). As presented, SNM is improved by 10% (increased from 338 mv to 372 mv). Consistent with our expectation, SNM variance improves by 23.5% (reduced from 17 mv to 13 mv). This shows that our proposed cell reduces cell stability sensitivity to process variations.

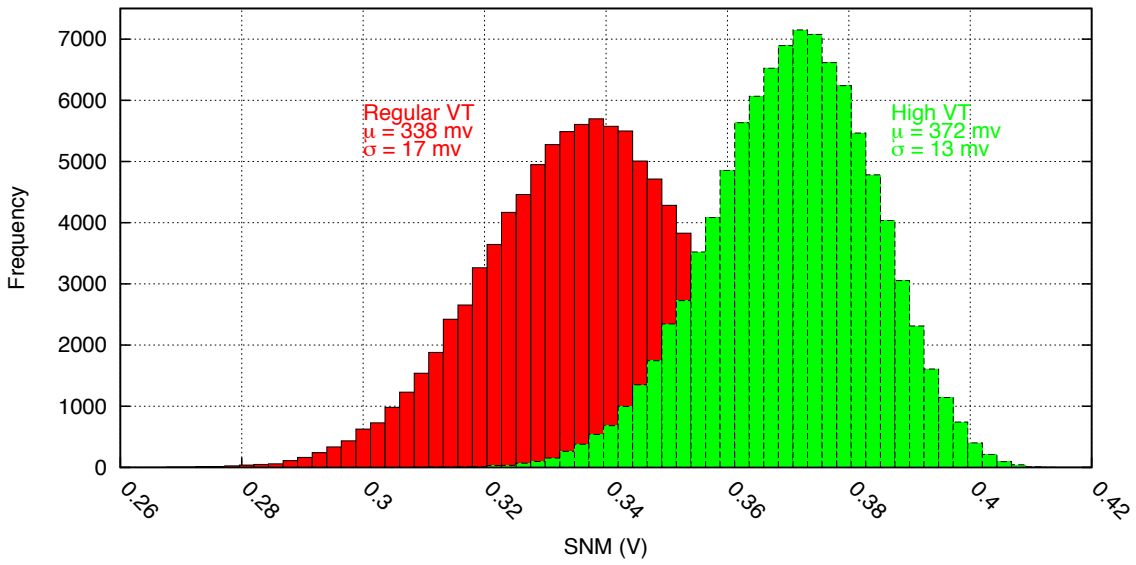

Figure 4.6 - SNM distribution of our proposed dual threshold 8T cell compared to the regular 8T cell

Similar to SNM, we report write trip point under process variations in Figure 4.7. In this figure, the distribution of write trip point is shown for our proposed cell (the left bell) compared to the regular 8T cell (the right bell). This figure shows that although write trip point voltage is reduced by 14% (from 420 mv to 360 mv) its variance is improved by 30% (reduces from 56 mv to 39 mv).

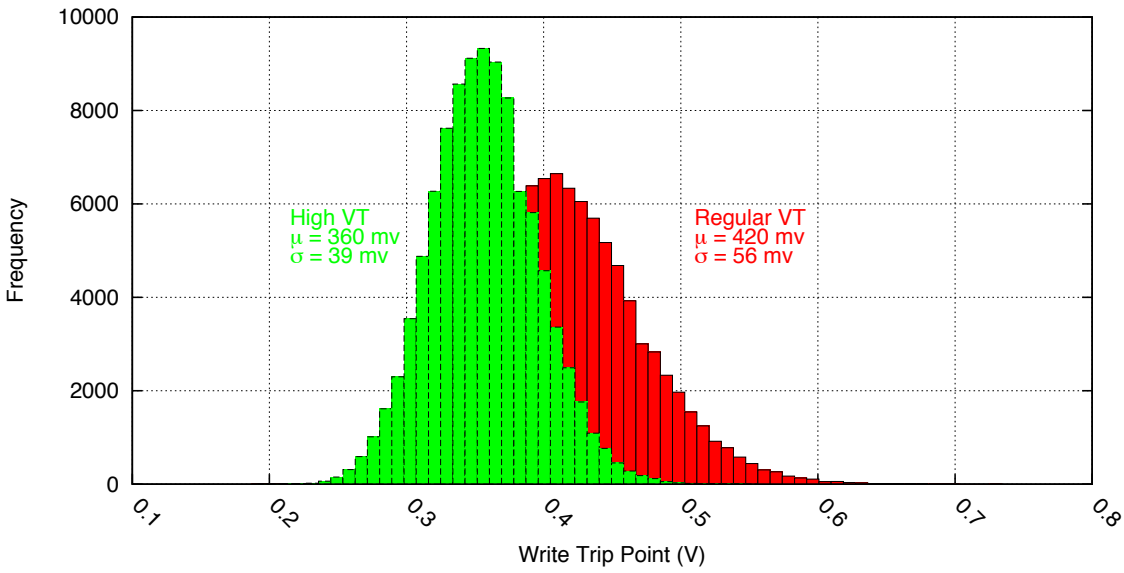

Figure 4.7 - Write trip point distribution of our proposed dual threshold 8T cell compared to the regular 8T cell

4.5 Conclusion

In this chapter, we showed that performance has low sensitivity to write latency. We also showed that exploiting our previously proposed techniques (WG and WG+RB) can reduce performance sensitivity to write latency effectively in 8T cells caches. Based on our observations, we introduced a low leakage 8T cell by exploiting dual threshold technique. Our proposed cell uses high threshold transistors only in the write operation path. Hence, it has no impact on read latency and only increases write latency. We reported performance slowdown due to the increase in write latency under optimistic and pessimistic scenarios. Moreover, we reported how our proposed cell improves leakage power and stability under process variations.

Chapter 5

Conclusion and Future Work

In this thesis, we examine power dissipation, performance, and stability concerns in L1 data caches. In Chapter 2, we discussed how leakage power dissipation has become a challenging issue in design of power efficient processors. We also showed that caches are an important contributor to leakage power dissipation due to accommodating a significant share of transistor budget. A previous study, drowsy cache [2], suggested using two voltage levels to power up caches to reduce leakage power with minimal performance overhead. Their proposed technique controls power supply voltage level at cache line-granularity. In this thesis, we presented that taking a word-granularity approach instead of line-granularity enhances the effectiveness of drowsy caches further as our simulation results confirmed that spatial locality in the studied benchmarks is low. Hence, word-granularity drowsy cache saves leakage power in words that are left in high leakage mode in line-granularity drowsy cache.

In Chapter 3 we showed that voltage scaling is limited by stability of caches, especially caches implemented using 6T cells. Hence, we focused on an alternative choice, 8T cell cache, which can improve stability by exploiting extra transistors. On the negative side, caches implemented using 8T cells suffer from column selection issue. We showed that a previous proposed solution, RMW [24], is very costly in terms of power and performance due to the required read operation before each write operation. We proposed two techniques, WG and WG+RB, to reduce RMW's overhead by exploiting a novel buffering mechanism. Our simulation results showed that our proposed techniques effectively reduce the imposed overhead by RMW.

Finally, in Chapter 4 we showed that performance is less sensitive to write latency compared to read latency. Based on this observation, we proposed a dual threshold 8T cell which improves leakage power dissipation at the cost of a modest increase in write latency. We showed that the imposed overhead due to increase in write latency can be hidden by exploiting WG and WG+RB effectively. We also showed that in addition to leakage power reduction, our proposed cell improves stability and robustness to process variations significantly.

5.1 Future Work

In this section, we provide some directions for future work on the proposed techniques in this thesis.

5.1.1 Adaptive Drowsy Cache

In Chapter 2, we showed that spatial locality in the studied benchmarks is low and for this reason the original drowsy cache misses significant leakage power reduction opportunities. Our proposed technique, ASL, enhances drowsy cache in applications with low spatial locality. Designing an adaptive drowsy cache which switches between line-drowsy cache and word-drowsy cache can be more effective. As this technique adjusts itself dynamically to the nature of locality of applications it can improve leakage power dissipation with minimizing performance penalty due to wakeup latency.

5.1.2 Multi-entry Set-Buffer

In Chapter 3, we studied 8T cells caches and proposed two techniques to improve RMW. Our techniques exploit a buffering mechanism which filters out a considerable amount of cache accesses, improving cache traffic and performance. The higher filtering rate, the higher cache traffic and performance improvement. One straight way to increase

filtering rate is exploiting a multi-entry Set-Buffer which allows buffering more than one cache set's data. However, a multi-entry Set-Buffer comes with area and power overhead. Therefore, finding the optimum balance between the power, performance, and area overhead can be considered as a future work.

Bibliography

[1] Hennessy, J. L., and Patterson, D. A.,: "Computer architecture: a quantitative approach," Morgan Kaufmann, 5th edition, 2011.

[2] Flautner, K., Kim, N. S., Martin, S., Blaauw, D., T. Mudge,: "Drowsy caches: simple techniques for reducing leakage power," Proceedings of the 29th annual international symposium on Computer architecture, 2002.

[3] Zhang, K., Bhattacharya, U., Chen, Z., et al.,: "SRAM design on 65-nm CMOS technology with dynamic sleep transistor for leakage reduction," IEEE J. Solid-State Circuits, vol. 40, no. 4, Apr. 2005, pp. 895-901.

[4] C. Kim and K. Roy,: "Dynamic V_t SRAM: a leakage tolerant cache memory for low voltage microprocessor," Proc. International Symposium on Low Power Electronics and Design, 2002, pp. 251–254.

[5] Kim, C. H. , Kim, J., Mukhopadhyay, S., Roy, K.,: "A forward body-biased low-leakage SRAM cache: device, circuit and architecture considerations," IEEE Trans. on Very Large Scale Integration Systems, vol. 13, no. 3, Mar. 2005, pp. 349-357.

[6] Azizi, N., Najm, F. N., and Moshovos, A.,: "Low-leakage asymmetric-cell SRAM," IEEE Transaction on Very Large Scale Integration (VLSI) Systems, 2003, pp. 701-715.

[7] Amelifard, B., Fallah, F., and Pedram, M.,: "Reducing the sub-threshold and gate-tunneling leakage of SRAM cells using dual-Vt and dual-Tox assignment," Proc. of Design, Automation and Test in Europe, 2006, pp. 1-6.

[8] Powell, M., Yang, S. H., Falsafi, B., Roy, K., and Vijaykumar, T. N.,: "Gated-V dd: a circuit technique to reduce leakage in deep-submicron cache memories," Proc. of International Symposium on Low Power Electronics and Design, 2000.

[9] Kaxiras, S., Hu, Z., and Martonosi, M.,: "Cache decay: exploiting generational behavior to reduce cache leakage power," Proc. of Int. Symposium on Computer Architecture, 2001, pp. 240-251.

[10] Kim, N. S., Flautner, K., Blaauw, D., Mudge, T.,: "Drowsy instruction caches—Leakage power reduction using dynamic voltage scaling and cache subbank prediction," in Proc. IEEE/ACM Int. Symp. Microarchitecture (MICRO-35), Nov. 2002, pp. 219–230.

[11] Shawkey, H. A., El-Dib, D. A. and Abid, Z.,: "Aggressive drowsy cache cells", International Journal of Electronics Vol. 97, No. 1, 2010, pp. 105–118

[12] Kim, N., Flautner, K., Blaauw, D., T. Mudge,: "Circuit and Microarchitectural Techniques for Reducing Cache Leakage Power," IEEE Transaction on Very Large Scale Integration (VLSI) Systems, VOL. 12, NO. 2, 2004,pp. 167-184.

[13] Jaydeep, K., Keejong P. K., Kaushik Roy,: "A 160 mV robust Schmitt trigger based subthreshold SRAM," Solid-State Circuits, IEEE Journal of 42.10 (2007): 2303-2313.

[14] Agarwal, A., Roy, K.,: "A noise tolerant cache design to reduce gate and sub-threshold leakage in the nanometer regime," Proc. Int. Symp. Low Power Electronics and Design, Aug. 2003, pp. 18–21.

[15] Zhou, H., Toburen, M. C., Rotenberg, E., Conte, T. M.,: "Adaptive mode-control: A static-power-efficient cache design," Proc. of Int. Conf. on Parallel Architectures and Compilation Techniques, 2001, pp. 61-70.

[16] Chang, Y., Lai, F., Yang, C.,: "Zero-aware asymmetric SRAM cell for reducing cache power in writing zero," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2004.

[17] Chen, C. F., Yang, S., Falsafi, B., Moshovos, A.,: "Accurate and Complexity Effective Spatial Pattern Prediction," Proc. of the 10th Intl. Symposium on High-Performance Computer Architecture, Feb. 2004.

[18] Burger, D. C., Austin, T. M., Bennett. S.,: "Evaluating Future Microprocessors-the SimpleScalar Tool Set," UW Computer Sciences Technical Report 1308, July, 1996.

[19] Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T., Brown, R. B.,: "MiBench: A free, commercially representative embedded benchmark suite," Proc. of Workshop Workload Characterization, 2001.

[20] Muralimanohar, N., Balasubramonian, R. and Jouppi. N. P.: 'CACTI 6.0: A tool to model large caches.' HP Laboratories, 2009.

[21] Agarwal, A., Li, H., Roy, K.,: "DRG-cache: a data retention gated-ground cache for low power," Proc. of the 39th annual Design Automation Conference, 2002.

[22] Wieckowski, M., Margala, M.,: "A novel five-transistor (5T) sram cell for high performance cache," Proc. System On Chips Conference, 2005.

[23] Tseng Y. H., Zhang, Y., Okamura, L., Yoshihara, T.,: "A new 7-transistor SRAM cell design with high read stability," Proc. IEEE International Conference on Electronics Design, Systems and Applications, 2010.

[24] Chang, L., Fried, D. M., Hergenrother, J., Sleight, J. W., Dennard, R. H., Montoye, R. K., Sekaric, L., McNab, S. J., Topol, A. W., Adams, C. D., Guarini, K. W., Haensch, W.,: "Stable SRAM cell design for the 32 nm node and beyond," IEEE Symposium on VLSI Technology Digest of Technical Papers, 2005.

[25] Lin, S., Kim, Y. B., Lombardi, F.,: "A highly-stable nanometer memory for low-power design," IEEE International Workshop on Design and Test of Nano Devices, Circuits and Systems, 2008.

[26] Highsmith, C. B., Chandrakasan, A. P.,: "A 256-kb 65-nm sub-threshold SRAM design for ultra-low-voltage operation," IEEE Journal of Solid-State Circuits, 2007.

[27] Verma, N., and Chandrakasan A. P.: "A 256 kb 65 nm 8T subthreshold SRAM employing sense-amplifier redundancy." IEEE Journal of Solid-State Circuits, 2008, pp.141-149.

[28] Mukhopadhyay, S., Mahmoodi, H., and Roy, K.: "Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2005, 24, (12), pp. 1859-1880.

[29] Morita, Y., Fujiwara, H., Noguchi, H., Iguchi, Y., Nii, K., Kawaguchi, H., and Yoshimoto. M.: "An area-conscious low-voltage-oriented 8T-SRAM design under DVS environment." IEEE Symposium on VLSI Circuits, 2007, pp. 256-257.

[30] Ranganathan, A.,: "Experimental Analysis of Snoop Filters for MPSoC Embedded Systems," Master Project, Ecole Polytechnique Federale de Lausanne, Switzerland, 2010.

[31] Sohi, G. S., Franklin, M.,: "High-bandwidth data memory systems for superscalar processors," Proc. International Conference on Architectural Support for Programming Languages and Operating Systems, 1991.

[32] Zhang, K., Bhattacharya, U., Chen, Z., Hamzaoglu, F., Murray, D., Vallepalli, N., Wang, Y., Zheng, B., Bohr M.,: "A 3-GHz 70MB SRAM in 65nm CMOS technology with integrated column-based dynamic power supply," IEEE International Conference Solid-State Circuits, 2005.

[33] Chang, L., Montoye, R. K., Nakamura, Y., Batson, K. A., Eickemeyer, R. J., Dennard, R. H., Haensch, W., and Jamsek. D.: "An 8T-SRAM for variability tolerance and low-voltage operation in high-performance caches." IEEE Journal of Solid-State Circuits, 2008, pp. 956-963.

[34] Flautner, K., Reinhardt, S., and Mudge, T.: "Automatic performance setting for dynamic voltage scaling." Proc. of Int. Conference on Mobile computing and networking, 2001, pp. 260-271.

[35] Ma, D., and Bondade, R.: "Enabling power-efficient DVFS operations on silicon." IEEE Circuits and Systems Magazine, 2010, 10, (1), pp. 14-30.

[36] Miftakhutdinov, R., Ebrahimi, E., and Patt, Y. N.: "Predicting performance impact of DVFS for realistic memory systems." Proc. of Int. Symposium on Microarchitecture, 2012, pp. 155-165.

[37] Park, S. P., Kim, S. Y., Lee, D., Kim, J, Griffin, W. P., and Roy. K.: "Column-selection-enabled 8T SRAM array with ~1R/1W multi-port operation for DVFS-enabled processors." Proc. of Int. Symposium on Low Power Electronics and Design , 2011, pp. 303-308.

[38] Nakagome, Y., Horiguchi, M., Kawahara, T., and Itoh K.: "Review and future prospects of low-voltage RAM circuits," IBM Journal of Research and Development, 2003, pp. 525-552.

[39] Kim, D., Chandra, V., Aitken, R., Blaauw, D., and Sylvester, D.: "Variation-aware static and dynamic writability analysis for voltage-scaled bit-interleaved 8-T SRAMs." Proc. of Int. Symposium on Low Power Electronics and Design, 2011, pp. 145-150.

[40] Lee, J., Xie, L., and Davoodi, A.,: "A Dual-Vt low leakage SRAM array robust to process variations," Proc. of Int. Symposium on Circuits and Systems, pp. 580-583.

[41] Farahani, M., and Baniasadi, A., "Performance and power solutions for caches using 8T SRAM cells." Proc. of Int. Symposium on Microarchitecture Workshops, 2012, pp. 74-80.

[42] Kim, D., Chandra, V., Aitken, R., Blaauw, D., and Sylvester. D.: "An adaptive write word-line pulse width and voltage modulation architecture for bit-interleaved 8T SRAMs." Proc. of Int. Symposium on Low Power Electronics and Design, 2012, pp. 91-96.

[43] Henning, J. L. 'SPEC CPU2006 benchmark descriptions.' ACM SIGARCH Computer Architecture, 2006, pp. 1-17.

[44] Lepak, K. M., and Lipasti, M. H.: 'Silent stores for free.' Proc. Of Int. Symposium on Microarchitecture, 2000, pp. 22-31.

[45] Kundinger, C. J., Lindberg, N. D., and Stec, E. J.: 'Eliminating silent store invalidation propagation in shared memory cache coherency protocols.' U.S. Patent 8,127,083, issued February 2012.

[46] Patel, A, Afram, F., Chen, S., and Ghose. K.: 'MARSS: A full system simulator for multicore x86 CPUs.' Proc. of Design Automation Conference, 2011. pp. 1050-1055.

[47] Karn, T., Rawat, S., Kirkpatrick, D., Roy, R., Spirakis, G.S., Sherwani, N., and Peterson, C.,: 'EDA challenges facing future microprocessor design,' IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.19, no.12, pp.1498,1506, Dec 2000.

[48] Farahani, M., Eslami, F., and Baniasadi, A.,: "Application specific low leakage data cache for embedded processors," Proc. of the Int. Green Computing Conference, 2013.

[49] Wang, J., Nalam, S., and Calhoun, B. H.,: "Analyzing static and dynamic write margin for nanometer SRAMs," Proc. of Int. Symposium on Low Power Electronics and Design, 2008, pp. 129-134.

[50] Grossar, E., Stucchi, M., Maex, K., Dehaene, W.: "Read stability and write-ability analysis of SRAM cells for nanometer technologies," IEEE Journal of Solid-State Circuits, 2006, pp. 2577-2588.

[51] Kao, J. T., Chandrakasan, A. P.,: "Dual-threshold voltage techniques for low-power digital circuits," IEEE Journal of Solid-State Circuits, 2000, pp. 1009-1018.

[52] Seevinck, E., List, F. J., Lohstroh, J.,: "Static-noise margin analysis of MOS SRAM cells," IEEE Journal of Solid-State Circuits, 1987, pp. 748-754.

[53] Rabaey, J. M., Chandrakasan, A. P., Nikolic, B.,: "Digital integrated circuits," Vol. 2. Englewood Cliffs: Prentice hall, 2002.

[54] Liu, H., Ferdman, M., Huh, J., Burger, D.,: "Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency," Proc. of the 41st annual IEEE/ACM International Symposium on Microarchitecture. 2008.