A Survey of Architectural Techniques For Improving Cache Power Efficiency

Sparsh Mittal

A Survey of Architectural Techniques For Improving Cache Power Efficiency

Sparsh Mittal
Future Technologies Group
Oak Ridge National Laboratory (ORNL), USA
sparsh0mittal@gmail.com

***Abstract*—Modern processors are using increasingly larger sized on-chip caches. Also, with each CMOS technology generation, there has been a significant increase in their leakage energy consumption. For this reason, cache power management has become a crucial research issue in modern processor design. To address this challenge and also meet the goals of sustainable computing, researchers have proposed several techniques for improving energy efficiency of cache architectures. This paper surveys recent architectural techniques for improving cache power efficiency and also presents a classification of these techniques based on their characteristics. For providing an application perspective, this paper also reviews several real-world processor chips that employ cache energy saving techniques. The aim of this survey is to enable engineers and researchers to get insights into the techniques for improving cache power efficiency and motivate them to invent novel solutions for enabling low-power operation of caches.**

***Index Terms*—Cache energy saving techniques, architectural techniques, dynamic energy, leakage energy, power management, low-power, energy efficiency, green computing, classification, survey**

I. Introduction

As we are entering into an era of green computing, the design of energy efficient IT solutions has become a topic of paramount importance [1]. Recently, the primary objective in chip design has been shifting from achieving highest peak performance to achieving highest performance-energy efficiency. Achieving energy efficiency is important in the design of all range of processors, such as battery-driven portable devices, desktop or server processors to supercomputers.

To meet the dual and often conflicting goals of achieving best possible performance and best energy efficiency, several researchers have proposed architectural techniques for different components of the processor, such as processor core, caches, DRAM (dynamic random access memory) etc. For several reasons, managing energy consumption of caches is a crucial issue in modern processor design. With each CMOS (complementary metal oxide semiconductor) technology generation, there is a significant increase in the leakage energy consumption [2], [3]. According to the estimates of International Technology Roadmap for Semiconductors (ITRS); with technology scaling, leakage power consumption will become a major industry crisis, threatening the survival of CMOS technology itself [4]. Further, the number of processor cores on a single chip has greatly increased over years and future chips are expected to have much larger number of cores [5]. Finally, to bridge the gap between the speed of processor and main memory, modern processors are using caches of increasingly larger sizes. For example, modern desktop processors generally have 8 MB last level caches [6] while server systems are designed with 24 to 32 MB last level caches [7], [8]. On chip caches consume 16% of the total power in Alpha 21264 and 30% of the total power in StrongARM processor core [9]. In both Niagara and Niagara-2 processors, L2 cache consumes nearly 24% of the total power consumption [10]. Thus, the power consumption of caches is becoming a large fraction of processor power consumption. To address the challenges posed by these design trends, a substantial amount of research has been directed towards achieving energy efficiency in caches. The focus of this paper is to review some of these techniques.

The rest of the paper is organized as follows. Section II presents a brief background on modeling of CMOS energy consumption and discusses the essential design principles which guide cache energy saving approaches. Section III and IV discuss the approaches which have been proposed for saving dynamic and leakage energy, respectively. In these sections, we first discuss the landscape of the techniques proposed and then discuss a few of them in more detail. We highlight the basis of similarities and differences between the techniques. Since leakage energy is becoming an increasing fraction of cache power consumption [2], [3], [59], we focus more on leakage energy saving techniques than dynamic energy saving techniques. Section V discusses the approaches which are used for saving both dynamic and leakage energy. To show real-life implementation of the research ideas, Section VI discusses some of the commercial chip designs which employ cache energy saving techniques. Finally Section VII concludes the paper.

As it is not possible to cover all the techniques in full detail in a review of this length, we take the following approach to restrict the scope of the paper. Although the techniques designed to improve performance are also likely to save energy, in this survey we only consider those techniques which aim to optimize energy efficiency and have been shown to improve energy efficiency. Further, cache energy saving can also be achieved using circuit-level techniques (e.g., low-leakage devices), however, in this paper we mainly focus on architecture-level techniques which allow runtime cache power management. Lastly, since different techniques have been evaluated using different simulation infrastructure and workloads, we do not include their quantitative improvement

Published in Elsevier Sustainable Computing: Informatics and Systems, 2013. DOI:10.1016/j.suscom.2013.11.001

TABLE I
CLASSIFICATION OF TECHNIQUES PROPOSED FOR DIFFERENT CACHES

Caches/Criterion	Energy-saving techniques (ESTs)
For first-level cache (L1)	[11]–[18]
For last-level caches (L2 or L3)	[19]–[30]
For instruction caches	[31]–[44]
For data caches	[9], [15], [16], [18], [18], [45]–[48]
ESTs utilizing hardware support	[12], [13], [48]–[53]
ESTs utilizing software support	[17], [23]–[26], [38], [54]–[56]
ESTs utilizing compiler support	[40], [41], [45], [57], [58]

results. Rather, we focus on their key design principles, which can provide valuable insights.

II. BACKGROUND AND RELATED WORK

The power consumption of CMOS circuits is mainly classified in two parts, namely dynamic power (also called active power) and leakage power (also called static power). In what follows, we present the modeling equations for both dynamic and leakage power in their simplified forms, which helps to gain insights into how energy saving techniques work. For a more detailed modeling, we refer the reader to [60], [61].

Dynamic power ($P_{dynamic}$) is dissipated whenever transistors switch to change the voltage in a particular node, while leakage power ($P_{leakage}$) is dissipated due to leakage currents that flow even when the device is inactive. Mathematically, they are given by,

$$P_{dynamic} = \alpha \times C_{eff} \times V_{DD}^2 \times F \quad (1)$$

$$P_{leakage} = V_{DD} \times I_{leak} \times N \times k_{design} \quad (2)$$

Here α shows the activity factor, V_{DD} shows the supply voltage, C_{eff} shows the effective capacitance and F shows the operating frequency. Further, N is the number of transistors, k_{design} is the design dependent parameter and I_{leak} is the leakage current, which is a technology dependent parameter.

From Eq. 1 we infer that, for a given CMOS technology generation, dynamic power consumption can be reduced by adjusting voltage and frequency of operation or by reducing the activity factor (e.g., by reducing the number of cache accesses or the number of bits accessed in each cache access etc.). Similarly, from Eq. 2, it is clear that, for a given CMOS technology generation, the opportunity of saving leakage energy lies in redesigning the circuit to use low-power cells, reducing the total number of transistors or putting some parts of caches into low (or zero) leakage mode. Based on these essential principles, several architectural techniques have been proposed, which we discuss in the next sections.

Modern processor cores have multi-level cache hierarchy with L1, L2, L3 caches etc. Also, typically at level one, data and instruction caches are designed as separate caches, while at lower levels (i.e., at level 2 and 3), instruction and data caches are unified. These caches have different properties and different techniques utilize these properties to save cache energy. Table I summarizes the techniques which are proposed for L1 or L2/L3, and for instruction or data caches. More detailed discussion of some of these techniques is presented in the following sections.

First-level caches (FLCs) are designed to minimize access latency, while last level caches (LLCs) are designed to minimize cache the miss-rate and the number of off-chip accesses. Accordingly, FLCs are smaller (e.g., 32KB or 16KB), have smaller associativity and employ parallel lookup of data and tag arrays [21]. In contrast, LLCs are much larger (e.g., 2MB, 4MB etc.), have higher associativity and employ serial (phased) lookup of data and tag arrays. Thus, as an example, an energy saving technique which increases the cache access latency is more suitable for LLCs, than for FLCs. Also, due to their relatively smaller sizes and large number of accesses, FLCs spend a larger fraction of their energy in the form of dynamic energy, while LLCs spend a larger fraction of their energy in the form of leakage energy [27], [30]. As for instruction/data caches, instruction access stream exhibits strong spatial and temporal locality and hence, the instruction cache is very sensitive to increase in access latency. The working set and reuse characteristic of instruction cache is different from that of data cache. Also, since instruction cache does not hold dirty data, reconfiguring it does not lead to write-back of dirty data and thus, reconfiguration of instruction cache can be more easily implemented than that of the data cache.

Table I also classifies the techniques based on whether they need hardware, software and/or compiler support. While compiler-based approaches incur no or minimal hardware overhead, compiler analysis may not be possible in all situations and also compiler only has limited information. Software-based approaches can leverage the software to make more complex decisions and consider the impact of energy saving techniques on components of the processor other than the cache, however, software-only approaches generally cannot exercise the opportunity of frequent reconfigurations and also incur larger implementation overhead than hardware-only approaches. Hardware-based approaches can utilize simple yet low-overhead algorithms, and can also exercise the opportunity of fine-grained and frequent reconfigurations. However, these approaches cannot easily take other components of processor into account.

Kaxiras and Martonosi [62] survey some of the architectural techniques proposed for saving energy in processors and memory hierarchies. This paper differs from their work, in that we review several recent developments which have been made in the fast evolving field of design of energy efficient architectural techniques. Also, we exclusively focus on the techniques aimed to save cache energy to provide more in-depth insights. Finally, to show the practical application of

the research ideas, we also discuss the examples of many commercial chips which use cache energy saving designs.

III. Dynamic Energy Saving Approaches

A. Overview

Recently several techniques have been proposed for saving dynamic energy. Before discussing them in detail, it is helpful to see their underlying similarities and differences by classifying them across several dimensions. Some techniques save dynamic energy by reducing the number of accesses to a particular level of cache hierarchy by using additional memory structures. These structures are used either for data storage [38], [55], [63]–[65], or *prediction* of cache access result [11], [14], [15], [49], [66] or for *pre-determination* of cache access result [18], [28], [50], [67]–[69].

Some techniques reduce the number of cache ways accessed in each cache access by either using software information [17] or compiler information [40], [57] or hardware-based approaches [12], [28], [47], [49], [50], [67], [69], [70]. A few techniques provision accessing frequent (hot) data with lower energy to reduce average dynamic energy of cache access [21], [46].

Some techniques trade-off access time for gaining energy efficiency by performing the various tasks required for cache accesses in sequential manner instead of simultaneously (i.e., in parallel manner). If a cache hit/miss decision has already been taken, further tasks can be avoided for saving dynamic energy. For example, a few techniques access cache ways sequentially [14], [37], [49], [54]; some other techniques perform matching of tag-bits in multi-step manner [71], while some techniques reduce the number of tag bits which are active or are required for comparison [22], [34], [72]–[74]. Techniques have also been proposed which reduce the data transferred in each cache access (e.g., [76]). Also, while the above mentioned techniques can also be extended for saving dynamic energy in multiprocessor systems, some techniques have been especially designed to address the issues related to saving energy in multiprocessor systems [57], [66], [75]. Table II provides an overview of dynamic energy saving techniques.

B. Discussion

Kin et al. [63] propose a small filter cache which is placed in front of the conventional L1 cache. By trying to serve most of the accesses from the data present in the filter cache, their technique reduces the number of L1 accesses and thus saves dynamic energy. The tradeoff involved in the use of filter cache is that for achieving a reasonably high hit-rate in the filter cache, the size of filter cache needs to be large, which, in turn, increases its access time and energy consumption.

Dropsho et al. [54] discuss an 'accounting cache' architecture, which works on the temporal locality property of caches. It uses LRU replacement policy which places most frequently used blocks near MRU way-positions and thus, most accesses are likely to hit in those ways. Using this idea, accounting cache logically divides cache ways into two parts, named primary and secondary. On any cache access, initially only the primary ways are accessed; the secondary ways are accessed only if there is a miss in primary ways. Thus, due to the reduction in the average number of way-comparisons, accounting cache saves dynamic energy. Udipi et al. [21] propose 'non-uniform power access' in the cache, where certain ways of the cache are accessible at low energy using low-swing wires. These ways are used as MRU ways. Thus, their technique saves energy by making the "common case" (i.e., hits to MRU ways) energy efficient.

Yang and Gupta [46] discuss a 'frequent value' data cache design, which divides the data cache into two arrays. For accesses made to frequent (i.e., hot) cache lines, only the first data array is accessed, while for accesses made to infrequent (i.e., cold) cache lines, both data arrays are accessed. Further, frequent cache lines are stored in encoded form and for accesses made to them, number of bit comparisons are reduced, leading to saving in dynamic energy.

Jones et al. [40] propose a technique for saving energy in instruction caches. Their technique works by using the compiler to place the most frequently executed instructions at the start of the binary. At runtime, these instructions are explicitly mapped to specific ways within the cache. When these way-placed instructions are fetched, only the specific ways are accessed. This leads to saving in cache dynamic energy.

Powell et al. [14] use the technique of predicting the cache way, which is likely to contain the data. On a cache-access, first only a single way is accessed. Thus, on a correct prediction, the cache behaves just like a direct mapped cache and the dynamic energy of cache access is reduced. However, on misprediction, all the cache ways have to be accessed. Thus, a misprediction leads to increase in cache access time and energy. Also, use of their technique leads to non-uniform cache hit latency on a right and wrong prediction of cache access result. To address this, way-selection based techniques have been proposed (see Section V).

Zhu and Zhang [49] propose a technique which combines way-prediction and phased access mechanisms to reduce dynamic energy of caches. Their technique uses the way-prediction mode to handle a cache hit and the phased mode (i.e., accessing tag array first and based on the result, accessing data array) to handle a cache miss. Their technique uses simple predictors to predict the result of cache access. When the access is predicted to hit, the way-prediction scheme determines the desired way and probes that way only. When the access is predicted to miss, the phased-access scheme accesses all the tags of the cache-set first and then only the appropriate cache way is accessed. Thus, their technique saves cache energy by reducing the number of accesses to data arrays.

Tsai and Chen [64] propose a technique for improving energy efficiency of embedded processors by using a memory structure called "Trace Reuse cache". The Trace Reuse cache is used at the same level of memory hierarchy as conventional instruction cache. It works by reusing the retired instructions from the pipeline back-end of a processor to efficiently deliver instructions in the form of traces. This enables the processor to sustain a higher instruction rate, which improves both performance and energy efficiency.

TABLE II
CLASSIFICATION OF DYNAMIC ENERGY SAVING TECHNIQUES

Criterion	Energy-saving techniques (ESTs)
ESTs using extra memory storage	for data storage [38], [55], [63]–[65], for *prediction* of cache access result [11], [14], [15], [49], [66] for *pre-determination* of cache access result [18], [28], [50], [67]–[69]
Reducing number of ways consulted in each access	using software [17], compiler [40], [57] , hardware [12], [28], [47], [49], [50], [67], [69], [70]
Reducing switching activity	Sequential cache-way access [14], [37], [49], [54], multi-step tag-bit matching [71], reducing active tag bits or those actually compared [22], [34], [72]–[74] accessing frequent (hot) data with lower energy [21], [46]
ESTs for multicores or multiprocessors	[57], [66], [75]

Ghosh et al. [67] propose a technique named 'Way Guard' to save dynamic energy in caches. This technique uses a segmented counting Bloom filter [77] with each cache way. By accessing this structure before a cache access, the cache controller can determine whether the requested cacheline is not present in a particular way. To save energy, the lookup of those cache ways can be avoided which do not contain the requested data.

Park et al. [71] discuss techniques for reducing tag comparisons by making an early estimation of cache hit or miss and using the result for skipping tag comparisons, if possible. Their method tracks the hotness or coldness (depending on the frequency of accesses made) of a cache line and for cold cache lines, a partial tag comparison is performed before full tag comparison to explore the possibility of early termination of tag comparison to save energy. Additionally, their method first compares the tags of hot lines, and only in case of a miss, it compares the tags of cold lines. Since by temporal locality property of caches, most of the cache hits are likely to occur in hot blocks, this method saves energy by reducing tag comparisons.

Kwak and Jeon [72] propose a technique to reduce the power consumed in tag-accesses. Their technique works on the observation that since program applications typically exhibit high memory access locality, most of the tag bits of successive memory addresses are expected to be same, except for a few differences in the LSB-side bits. Thus, by storing the MSB-side bits in compressed form, the actual number of bits required for comparison can be reduced. Based on this, their technique logically divides the cache tag in two parts, namely lower tag bits and higher tag bits. For applications with low memory access locality, the number of bits which are taken as lower tag bits (i.e., LSB-side bits) need to be larger and vice versa. Further, the higher tag bits are stored in compressed form. On any cache access, when tag-matching is performed; all the lower tag bits and only the compressed higher tag bits are compared, which leads to saving of cache energy.

Guo et al. [58] present techniques to reduce the energy overhead of prefetching techniques. One of their techniques uses compiler information to identify memory addresses which are useful for prefetching and then, prefetches only those memory addresses. Another technique proposed by them uses a small prefetch filter buffer (PFB) to reduce the overhead of L1 tag look-up related to prefetching. PFB is used to store the most recently prefetched cache tags and a prefetching address is checked against PFB. If the address is found in the PFB, prefetching is avoided, which leads to saving of energy.

Fang et al. [17] propose a method to save dynamic energy by utilizing software semantics in cache design. For saving instruction cache energy, their technique works on the observation that a user-mode instruction fetch cannot hit in a cache line that contains kernel-mode code. Based on this, on a user-mode instruction fetch, only the cache-ways containing user-mode code are accessed. For saving data cache energy, their technique works on the observation that an access to stack data cannot hit in a way containing heap data and vice versa. Hence, they store an extra bit with each tag to identify whether the data belongs to stack or heap. Checking this bit at the time of access helps in early identification of cross-checks between stack and heap. Using this information, checking those ways can be avoided which are sure to lead to cache miss.

Recently, researchers have proposed 3D-stacked DRAM caches, where 3D die stacking is used to interconnect a processor die with a stack of DRAM dies using through-silicon vias (TSVs) [123]. 3D die stacking technology promises lower interconnect power consumption, smaller chip footprint and increased on-chip bandwidth and for these reasons, this technology is already finding commercial adoption [124]. However, 3D die stacking technology also presents significant power and thermal challenges [125]. To address this, several architecture-level techniques have been proposed.

Jevdjic et al. [126] present a 3D stacked DRAM cache design which aims to achieve high energy efficiency and performance by intelligent cache management. In DRAM caches, using a small (e.g. 64B) block size leads to optimized use of cache capacity and bandwidth, but high lookup latency. On the other hand, using a large granularity (e.g. 4KB pages) leads to fast lookup and reduced tag overhead at the cost of poor bandwidth and cache capacity utilization. Jevdjic et al. propose a cache design, which allocates data at granularity of pages, but fetches only those pages that will be accessed during page's residency in the cache. This avoids bringing unnecessary pages into the cache which improves cache and bandwidth utilization. The prediction of useful blocks is made by identifying spatial correlation. The experimental results show that in terms of energy efficiency, their technique outperforms conventional (i.e. without die-stacking) cache and also block-based and page-based 3D stacked DRAM cache designs.

TABLE III
CLASSIFICATION OF LEAKAGE ENERGY SAVING TECHNIQUES (RECONFIG.= RECONFIGURATION)

Criterion	Energy-saving techniques (ESTs)
Circuit-level ESTs	state-preserving [33], [48], [51], [78], state-destroying [44]
Micro-architectural ESTs	state-preserving [19], [31], [39], [42], [45], [52], [53], [79]
	state-destroying [13], [20], [23]–[25], [29], [43], [70], [80]–[86]
	either or both [30], [41], [87], [88]
Reconfig. granularity	way-level [20], [52], [53], [56], [89]–[93]
	set-level (or bank-level) [43], [92]
	hybrid (set and way) level [23], [81], [94]
	cache block-level [13], [29], [31], [41], [42], [45], [78]–[80], [84], [88]
	cache sub-block level [86], cache color level [24]–[26]
	cache sub-array level [82]
Reconfig. Frequency	Static [89], [91], [93], dynamic (all in the next three rows)
dynamic reconfig.	Fixed large interval [13], [20], [23]–[25], [42], [43], [52], [81], [82], [84], [92], [94]
	variable interval [29], [79], [80]
	continuous reconfig. [30], [31], [41], [45], [52], [53], [78], [86], [87]
Basic property on	Inclusion property of cache hierarchies [30]
which ESTs work	temporal locality [13], [29], [31], [41], [42], [45], [48], [52], [79], [80], [84], [86]
	varying working set size [23]–[26], [43], [81], [94], [95]
What is turned-off	Only data array (and not tag array) [29], [80], [96]
	both data and tag arrays (almost all others)
Profiling: offline	Offline (or compiler analysis) [41], [43], [45], [81], [93], [94], [97]–[100]
or online	online (almost all others)
Thermal aware ESTs	[88], [92], [93], [101], [102]
ESTs for multi-cores/	[26], [27], [56], [70], [85], [87], [96], [99], [100], [103]–[113]
multi-processors	
Integration with	DVFS [36], [90], [109], [114]–[116]
other approaches	data compression [83], [106], [117], [118], prefetching [119]
ESTs in different	QoS systems [24], [26], real-time systems [98], [99], [120]
application contexts	embedded multitasking systems [97], [99], [121], [122]

Sun et al. [127] propose a heterogeneous 3D DRAM architecture to implement both L2 cache and main memory within the 3D stacked DRAM dies. Because of larger density of DRAM, larger sized DRAM caches can be architected in the same area compared to SRAM caches. Sun et al. study use of multiple (viz. 2, 4 and 8) layers of stacked DRAM and show that their proposed 3D DRAM cache design offers better energy efficiency than 2D SRAM design. Moreover, use of larger number of layers helps in reducing the access latency and energy consumption.

IV. LEAKAGE ENERGY SAVING APPROACHES

A. Overview

As explained before, leakage energy saving approaches work by turning off a part of the cache to reduce the leakage energy consumption of the cache. Based on the data retentiveness of turned-off blocks, the leakage energy saving techniques are classified into two broad types, namely state-preserving and state-destroying techniques. The state-preserving techniques turn off a block while preserving its state (e.g., [33], [48], [51], [78]). This means that when the block is reactivated, it does not need to be fetched from next level of memory. State-destroying techniques (e.g., [44]) do not preserve the state of the turned-off block, but generally save more energy in the low-power states than the state-preserving techniques. Several microarchitecture level techniques utilize state-preserving leakage control (e.g., [19], [31], [39], [42], [45], [52], [53], [79]), while others employ state-destroying leakage control (e.g., [13], [20], [23]–[25], [29], [43], [70], [80]–[86]). Some researchers have proposed techniques which work with either of or both of state-preserving or state-destroying leakage control mechanism [30], [41], [87], [88]. Li et al. [128] compare the effectiveness of state-preserving and state-destroying techniques. They conclude that when the cost of fetching a missed block is high, state-destroying techniques incur a large penalty and thus, state-preserving techniques show superior performance. However, when the cost of fetching a missed block is not high, a state-destroying technique can be superior to the state-preserving technique, since a state-destroying technique completely turns off the block, thus saving more energy.

The energy saving techniques turn off cache at the granularity (unit) of certain cache space, such as a single way or a single block at a time. Based on this granularity, leakage energy saving techniques can be classified as way-level [20], [52], [53], [56], [89]–[93], set-level (or bank-level) [43], [92], hybrid (set and way) level [23], [81], [94], cache block-level [13], [29], [31], [41], [42], [45], [78]–[80], [84], [88], cache sub-block level [86], cache color level [24]–[26] or cache sub-array level [82] etc.

To demonstrate the typical values of the different cache parameters, we take the example of an 8-way set-associative cache of 2MB size with 64B block size and 8-byte sub-block. For computing number of cache colors, we assume that the system page size is 4KB. Then, the number of cache blocks is 32,768 and number of sub-blocks is 262144. The number of cache ways is 8 and the number of cache colors is 64. The number of sets is 4096, however, it is noteworthy that the selective-sets approach allocates cache only at granularity of power of two sets, such as 4096 or 2048 or 1024 sets etc. The cache sub-array level allocation approach in [82]

divides a 2MB cache into 6 heterogeneous parts (called sub-caches), which have the size of 1MB, 512KB, 256KB, 128KB, 64KB and 64KB. Thus, different techniques work at different granularities.

Reconfiguration at each of these levels of granularities provides its own advantages and disadvantages. Unlike selective-sets and cache-coloring approachs, selective-way approach does not require change in the set-decoding on cache reconfiguration, which leads to smaller reconfiguration overhead and easier implementation. However, selective-way approach harms the associativity of the cache; for example, turning-off all but one way of a last-level cache turns it into a direct-mapped cache which leads to very high miss-rate and off-chip accesses. Also, selective-way approach provides limited granularity, which is at most equal to the number of ways. To achieve high granularity with selective-ways approach requires use of highly-associative caches, which also have high access time and energy. Selective-sets approach can potentially provide large granularity, however, in practice, it is observed that reducing the cache size below 1/8 or 1/16 significantly increases the miss-rate [23], [43]. Cache coloring can provide better granularity and smaller reconfiguration overhead than selective-sets approach (by using mapping-table [24], [25]), however it also has higher implementation overhead. Hybrid (selective-sets and selective-ways) approach aims to provide higher granularity than either of the two approaches and combine their benefits; however its implementation overhead is higher than either of the selective-sets or selective-ways approach alone. Cache block-level reconfiguration provides much higher granularity than any of the above mentioned approaches and does not change set-decoding on reconfiguration. This approach typically makes decision to turn-off each block locally based on the access/miss characteristics of each block. Cache sub-block level reconfiguration provides the highest granularity, however, it incurs high implementation overhead. Also note that with increasing granularity, the reconfiguration decision logic generally becomes increasingly complex. Moreover, increased granularity does not necessarily provide higher energy savings if the application does not benefit from it.

From the point of view of the time-interval at which reconfiguration is done, the techniques can be classified[1] based on whether they use a fixed (static) configuration throughout the execution (e.g., [89], [91], [93]) or use dynamic reconfiguration (i.e., the configuration is dynamically changed during the execution). Among techniques using dynamic reconfiguration, some techniques switch cache blocks at the boundary of a fixed, large interval size [13], [20], [23]–[25], [42], [43], [52], [81], [82], [84], [92], [94]; some techniques use variable interval size (time length) [29], [79], [80], some techniques switch cache blocks throughout the execution (for example, before or after cache accesses) [30], [31], [41], [45], [52], [53], [78], [86], [87].

To achieve cache energy saving, different techniques utilize different properties of the caches. Some techniques save energy by exploiting inclusion property of cache hierarchies [30] while some other techniques exploit generational nature or temporal locality property of caches [13], [29], [31], [41], [42], [45], [48], [52], [79], [80], [84], [86]. A few other techniques dynamically reconfigure the cache based on the working set size[2] size of applications and turnoff the rest of the cache to save energy [23]–[26], [43], [81], [94], [95]. In multicore systems, this approach extends to partitioning the cache between different applications and turning off rest of the cache for saving cache power [56].

Leakage energy saving techniques can also be classified depending on whether they turnoff only data array or both data and tag arrays. For example, a few techniques turnoff only data arrays of unused sections and keep the tag arrays turned-on to guide their algorithms [29], [80], [96], while most other techniques afford to turnoff both data and tag arrays of unused sections. For guiding their algorithms, some techniques require offline profiling or compiler analysis for their function (e.g., [41], [43], [45], [81], [93], [94], [97]–[100]), while most others use only runtime information for guiding their algorithms.

Since leakage energy varies exponentially with the temperature, an increase in chip temperature increases the leakage energy dissipation in caches, which, in turn, further increases the chip temperature. To take chip temperature into account while modeling and minimizing leakage energy, several techniques have been proposed [88], [92], [93], [101], [102]. Such techniques are referred to as thermal-aware or thermal-sensitive techniques. Also, while many of the above mentioned techniques can be extended to multicore or multiprocessor systems, several techniques have been especially designed to address the issues arising in multicore or multiprocessor systems [26], [27], [56], [70], [85], [87], [96], [99], [100], [103]–[113].

To achieve additional amount of energy saving, cache energy saving techniques have been synergistically integrated with some other approaches, such as DVFS (dynamic voltage/frequency scaling) [36], [90], [109], [114]–[116], data compression [83], [106], [117], [118], prefetching [119], etc. Further, cache leakage energy saving has also been discussed in the context of QoS (quality-of-service) systems [24], [26], real-time systems [98], [99], [120] and embedded multitasking systems [97], [99], [121], [122] etc. Table III provides the overview of leakage energy saving techniques.

B. Discussion

For both state-preserving and state-destroying leakage control, architectural techniques make use of some well-known circuit-level mechanisms. Powell et al. [44] propose a circuit design named 'gated V_{dd}', which facilitates state-destroying leakage control. This technique adds an extra transistor in the supply voltage path or ground path of the SRAM (static random access memory) cell. For reducing the leakage energy of the SRAM cell, this transistor is turned off and by stacking effect of the transistor, the leakage current is reduced by orders of magnitude. Similarly, Flautner et al. [48] discuss a

[1]Some techniques have multiple variants with different characteristics and hence, they are classified in multiple groups.

[2]Working set of an application is the number of unique cache lines accessed during a given execution interval.

circuit design named 'drowsy-cache', which facilitates state-preserving leakage control. This technique uses two voltage supplies to the cache, one of which is low voltage and the other is high voltage. For reducing the leakage energy of the SRAM cell, the cache controller switches the operating voltage of the cell to low voltage, thus putting the cell in low-leakage mode. When this line is accessed the next time, the supply voltage is again switched to high, thus the cache-block consumes normal power. Kim et al. [33] propose a "super-drowsy" circuit design and Agarwal et al. [78] propose a gated-ground circuit design, both of which behave similar to the drowsy cache, except that they only require a single voltage supply. Similarly, another state-preserving circuit design, named multithreshold CMOS (MTCMOS), dynamically changes the threshold voltage of the SRAM cell by modulating the backgate bias voltage to transition the cell to low-leakage mode [129].

Several energy saving techniques are based on the generational nature of cache access, which implies that cache lines have a period of frequent use when they are first brought into the cache, and then have a period of "dead time" before they are evicted. So, if a cache line has not been accessed for a certain number of cycles (called 'decay interval' or 'update window'), it indicates that the line has become dead and it can be put in low leakage mode for saving energy. Using this principle, Flautner et al. [48] propose 'drowsy-cache' technique which puts the dead cache lines into low-power *state-preserving* mode. Similarly, Kaxiras et al. [13] propose 'decay cache' technique which puts the dead cache lines into low-power *state-destroying* mode.

Several researchers have proposed improvements to the original decay-cache technique. Since the optimal value of the decay interval varies with the applications, Zhou et al. [80] propose a technique for dynamically adapting decay interval for each application. Their technique only turns off data and keeps tags alive. Using tags, their technique estimates the hypothetical miss rate, which would be there if all the data lines were active. Then, the aggressiveness of cache line turning off is controlled to make the the actual miss rate closely track the hypothetical miss rate. Abella et al. [29] keep track of the interaccess time and the number of accesses for each cache line and use this to compute suitable decay time for each individual cache line.

Kadayif et al. [119] study the interaction of prefetching and cache line turning-off. The prefetching mechanism is used to improve the performance of the processor while the leakage control mechanism is used to save energy in caches of the processor. Thus, their work studies how these two techniques interact and proposes methods to enable their synergistic operation. Since normal cache lines and those brought by prefetching have different usage patterns, their technique works by using different decay intervals for both kinds of cache lines.

Petit et al. [52] use recency information of the set-associative structure of caches to keep either a single or two MRU way(s) alive and switch rest of the ways to drowsy mode. Since most accesses are likely to hit in MRU way(s), this technique saves energy while also improving the number of hits to alive (i.e., non-drowsy) cache lines. Bardine et al. [20] use the ratio of hits to the MRU way and that to the least recently used active way in all the sets to estimate the degree of locality present in the memory access stream. A high value of ratio indicates that most accesses hit near MRU way and hence, if more than two ways are enabled, a single cache way can be disabled using state-destroying leakage control mechanism. Conversely, a low value of the ratio indicates that cache hits are distributed over different ways and hence, a single cache way is enabled.

Zhao et al. [79] adapt the interval of transitioning the cache-blocks to drowsy mode by taking into account the reuse distance of the caches. The reuse distance of a memory access is defined as the number of distinct cache lines referenced since the last reference to the requested line. The reuse distance reflects the temporal locality of the access pattern. A small reuse distance indicates that there exists a strong likelihood of future reference and vice versa. Thus, instead of using a fixed interval based on the number of cycles, their technique transitions a cache block to drowsy mode after a fixed N distinct references to the block. Thus, their technique also improves the number of hits to the alive cache lines.

Mohyuddin et al. [53] propose a technique for saving leakage energy by maintaining different ways of a cache at different state-preserving power saving modes depending on their replacement priorities. Going from the MRU way to the LRU way, cache lines are kept in increasingly aggressive power saving mode which also have increasingly larger penalties of cache line wakeup.

Chung and Skadron [31] adapt the drowsy cache technique for instruction caches using branch predictor information. On an access to the cache-block, the drowsy cache technique incurs the wakeup penalty, which lies at the critical access path. To hide this latency, Chung et al. propose using the branch-predictor to identify the next cache-block which would be accessed. Based on this, *only* the desired cache-block can be woken up before the actual access. In the case of branch mispredictions, the prediction of cache-block also becomes wrong, however, in such cases, the extra wakeup time is hidden due to the time taken in misprediction recovery. Zhang et al. [45] propose a compiler technique for saving cache leakage energy. This technique uses the compiler to perform program data reuse analysis to determine the cache access pattern. Using this information, all the cache lines are placed into state-preserving low-leakage mode and a cache line is brought to normal power-mode just before it is accessed.

Since different programs and even different phases of an application have different cache requirement, several techniques save cache energy by dynamically reconfiguring the cache for each program or program phase and turning off the rest of the cache. Using this idea, Albonesi [89] proposes *selective-ways* approach where some of the ways of the cache are turned off to save energy. Yang et al. [43] discuss *selective-sets* approach where leakage energy is saved by turning off some of the sets of the cache. Yang et al. [94] also discuss *selective sets and ways* where both the number of sets and ways can be altered to save leakage energy in data and instruction caches.

To dynamically reconfigure caches using the selective-ways approach, program response for different number of cache

ways needs to be estimated. For this purpose, researchers generally utilize utility monitors based on Mattson stack algorithm (e.g., [130], [131]). Similarly, for utilizing selective-sets approach, researchers generally use set-sampling method and multiple auxiliary tags for getting profiling information (e.g., [23]). Mittal et al. [23] present a hybrid set and way reconfiguration approach for leakage energy saving in last level caches. Their technique uses dynamic profiling for predicting cache usage and energy efficiency of the application under multiple cache configurations. Using these estimates, at the end of a fixed interval, the cache is reconfigured to the best configuration.

Li et al. [30] discuss different techniques for saving cache leakage energy by exploiting the data duplication across different levels of the cache hierarchy. Their technique works by putting an L2 block in low leakage (either state-preserving or state-destroying) mode, when the block also exists in the L1 cache. Their technique essentially tries to make the cache hierarchy non-inclusive for live cache lines.

Kotera et al. [56] use selective-ways technique in the context of chip multiprocessors to achieve both cache partitioning and energy saving. Their technique works by allocating just suitable number of cache ways to different programs and turning off the rest of the ways for saving cache energy.

Monchiero et al. [107] propose techniques to save leakage energy in private snoopy L2 caches of chip multi-processors (CMP) by selectively switching off the infrequently used lines. One of their technique turns off cache blocks which have been invalidated due to coherence protocol itself. The advantage of this technique is that it does not induce extra misses and hence does not incur a performance penalty. They also propose other techniques which work by carefully choosing coherence-state transitions, on which a block is decayed, so that the leakage energy is saved with minimal performance loss.

Reddy and Petrov [97] present an energy saving approach for embedded systems. They use an off-line algorithm to select the best cache partitioning for different running applications and use this information at runtime. They also show the usefulness of their technique in reducing inter-task interference in a preemptive multitasking environment. Similarly, Paul and Petrov [122] propose an approach for partitioning instruction cache for saving energy in embedded multitasking systems.

Since most energy saving techniques aim to aggressively save cache energy, use of them may lead to large performance degradation which may be unacceptable in QoS systems. Mittal, Zhang and Cao [24] present a technique for saving cache energy in QoS systems. Their technique allocates cache at granularity of cache colors. Using auxiliary tag structure for different cache configurations (having same number of ways and different set counts), their technique predicts the cache energy and program performance for multiple cache configurations. Using this, in each interval, the cache is dynamically reconfigured to a suitable cache size such that the QoS target of the program can be met, while saving maximum possible amount of energy.

Ku et al. [92] propose a thermal-aware leakage energy saving technique, which works on the intuition that apart from saving leakage energy in turned off cache lines, leakage energy of active parts can also be saved by intelligently turning off cache lines, such that chip temperature is reduced. Based on this, for a same amount of turned off cache; instead of turning off *entire* banks, their technique turns off *alternating* rows of memory cells in the bank. For the same number of turned off lines, compared to *thermal-unaware* schemes, their scheme increases the distance between active blocks, and thus reduces the chip temperature which also lowers the leakage energy dissipation.

Compared to CPUs, GPUs (Graphics Processing Units) typically use caches of smaller size, and hence, most of the work on cache energy saving has targeted CPU architecture. However, cache energy saving in GPUs has recently attracted the attention of researchers. Wang et al. [132] discuss a microarchitectural technique for saving energy in both L1 and L2 caches in GPUs. They propose putting the L1 cache in state-preserving, low-leakage mode when there are no threads ready to be scheduled. Further, the L2 cache is put in low-leakage mode when there is no memory request.

V. Approaches for Saving Both Dynamic and Leakage Energy

Several studies present reconfigurable cache architectures which offer flexibility to change one or more parameters of cache. By taking advantage of the flexibility offered by these architectures, both dynamic and leakage energy can be saved.

C. Zhang et al. [91] propose a highly-configurable cache architecture which contains four separate banks that can operate as four separate ways. By concatenating these ways, the associativity of the cache can be altered and/or some ways can be shut down. Thus, the associativity of the cache can be changed to either 1, 2 or 4. Similarly, by configuring the fetch unit to fetch different size of cache lines, the cache line size can also be altered. Wang and Mishra [98] and Rawlins and Gordon-Ross [106] use this architecture for saving cache energy. For example, Wang and Mishra profile several possible configurations of L1 data cache, L1 instruction cache and unified L2 cache in offline manner and at runtime, explore different possible combinations of two-level cache hierarchy to find the most energy efficient configuration. Similarly, Rawlins and Gordon-Ross discuss their technique for saving cache energy in heterogeneous dual-core systems by tuning L1 cache size, while addressing the issues presented by multicore operation such as core-interactions, data coherence etc.

Abella and González [133] propose a 'Heterogeneous Way-size cache' where the number of sets in each cache way can be different. The only requirement is that the number of sets in each way should be a power of two value. Note that the conventional caches use same number of sets in each cache way. By adapting the size of each way according to the application requirement, their technique saves both dynamic and leakage energy.

Benitez et al. [82] propose 'Amorphous cache' which uses heterogeneous sub-caches that can be selectively powered-off. Thus Amorphous cache allows changing the total cache size and/or set-associativity, depending upon the program requirement for saving both leakage and dynamic energy.

Wong et al. [134] propose using different voltages for different levels of cache. Unlike drowsy cache technique [48], their technique does not dynamically change the voltage of the cache block. Rather, throughout the execution, the cache is operated at a fixed voltage, which is lower than the core-voltage. Moreover, level two cache is operated at lower voltage than the level one cache, which, in turn, can operate at lower voltage than the core. At the interface between these two components, voltage level converters are used. Since the cache is operated at low voltage, both leakage and dynamic energy of access are saved.

Jiang et al. [105] propose a technique for saving energy in chip multiprocessors using asymmetric last-level caches. Their approach works by allocating suitable amount of cache to each application; however their approach differs from conventional approaches based on cache reconfiguration or partition (such as [43], [82] etc.) in that, asymmetric caches are physically separated private caches of different sizes and to use them for achieving energy efficiency requires smart scheduling techniques. Thus, their technique uses OS scheduler to assign applications with large working sets on large caches and those with smaller working sets on smaller caches. Smaller caches reduce access energy and operating voltage and larger caches use cache line turnoff to save leakage energy.

Alves et al. [86] propose a technique for saving cache leakage and dynamic energy. Their technique predicts the usage pattern of the sub-blocks of a cache block, which includes *which* sub-blocks of a cache line will be actually used and *how many times* it will be used. This information is used to bring only those sub-blocks in the cache which are necessary and turns them off after they have been touched the estimated number of times. Further, they augment the cache replacement policy to preferentially evict those cache blocks for which all sub-blocks have become dead. Note that compared to other techniques which utilize cache liveliness information (e.g., decay cache [13] or drowsy cache [48]) and work on cache block level, the technique proposed by Alves et al. works on cache sub-block level.

Several researchers have presented techniques for synergistically using both leakage and dynamic energy saving techniques. For example, Giorgi and Bennati [65] demonstrate that using filter cache [63] reduces the number of accesses to L1 cache, which, in turn, enables effectively using leakage energy saving techniques in L1 caches. Similarly, Keramidas et al. [50] propose a way-selection based technique for additionally saving dynamic energy in the caches which use decay-based leakage energy management. Their technique works on the observation that in a cache, using cache-decay mechanism [13] for saving leakage energy, several cache-blocks may be dead. Thus, by making an early determination of these dead blocks, the accesses to these cache blocks can be avoided, which leads to saving of dynamic energy of the cache. To this end, their technique uses a decaying Bloom filter to track liveliness information of each cache way of the cache sets. The Bloom filter enables an early prediction of cache miss, and thus, based on this information, only selected cache ways are accessed, which leads to saving of dynamic energy of cache access. Since way-selection mechanism, unlike way-prediction mechanism, gives definite information about a cache miss, it always leads to uniform cache hit latency.

VI. Cache Energy Saving in Real-world Chips

In this section, we discuss a few commercial chips which provide runtime power management features.

Malik et al. [135] discuss Motorola M Core M340 processor which provides the flexibility of turning-off ways of L1 cache for saving energy. Gerosa et al. [136] discuss the design of a low-power Intel processor designed for Mobile Internet Devices (MID) and Ultra-Mobile PCs. This chip uses energy saving techniques both in L1 and L2 caches. The L2 cache is 8-way, 512KB cache and for applications with low cache demand, up to 6 ways can be turned off using power-gating and sleep transistors, resulting in $10\times$ reduction in leakage power.

Chang et al. [137] discuss the design of 65-nm, 16 MB, on-die L3 cache for dual core Intel Xeon 7100 chip. This cache implements low-power techniques to save both leakage and dynamic energy. To save leakage energy, state preserving techniques are used in the SRAM array and peripherals which reduce the cache leakage by more than 2X over an unoptimized cache. To save dynamic energy, at each cache access, only 0.8% of all array blocks are powered up. Similarly Intel's 45nm 8-core Enterprise Xeon processor [138] and 32nm Westmere processor series [139] also implement several design features for saving leakage and dynamic energy in caches.

Sakran et al. [140] discuss the architecture of 65nm dual-core Intel Merom processor chip which has 4MB shared L2 cache. This chip uses sleep transistors (STs) in memory arrays, decoders and write drivers. Using STs, the leakage is reduced by 3 times while still preserving the data. Further, the chip uses microarchitectural techniques to identify low usage of the cache and then switches the STs of some parts of the cache to shut-off mode which reduces the array leakage by 7 times.

Gammie et al. [3] discuss 'SmartReflex' power management technology used by Texas Instruments mobile processors, such as 90nm OMAP2420 processor [141], 65 nm OMAP3430 processor [142] and the 45 nm 3.5 G Baseband and Multimedia Application Processor [143]. For saving both dynamic and leakage energy in SRAM, these processors use techniques such as state-preserving and state-destroying leakage control, voltage scaling etc.

George et al. [144] discuss the architecture of Intel 45-nm dual-core chip, codenamed Penryn, which has several features for saving both leakage and dynamic energy. Penryn is based on Core microarchitecture and has a unified, 24-way L2 cache having a size of 6MB. The cache is organized in 1MB slices each containing 4,096 lines and 4-ways. Also, each slice consists 32 data banks, each of which contains 2 sub-arrays. For saving leakage energy, the hardware design allows turning off cache at the granularity of a single slice (4 ways). For saving dynamic energy, the cache controller activates only half of the sub-array for any L2 access.

Branover et al. [145] discuss the architecture of AMD Fusion APU (accelerated processing unit), named Llano which is designed with 32-nm technology. The Llano APU consists

of four x86 CPU cores each of which has a private 1MB L2 cache. For leakage energy saving, Llano provisions power gating each core and its associated L2 cache separately.

Zhang et al. [146] discuss the design of 65nm SRAM which uses sleep transistors for saving leakage energy. This SRAM can dynamically control sleep transistors to reduce leakage energy of the cell by 3 to 5 times, while preserving its information content. Several other SRAM designs implement power management features, for example, [147]–[150].

VII. Concluding Remarks

Driven by continuous innovations in CMOS fabrication technology, recent years have witnessed wide-spread use of multicore processors and large sized on-chip caches for achieving high performance. However, due to this, total power consumption of processors is rapidly approaching the "power-wall" imposed by thermal limitations of cooling solutions and power delivery. Thus, to be able to continue achieving higher performance using technological scaling, managing the power consumption of processors has become a vital necessity.

In this paper, we have reviewed several architectural techniques proposed for managing dynamic and leakage power in caches. We have also discussed examples of commercial chips, which provide mechanisms to save cache power at runtime. We believe that our survey will enable researchers and engineers to understand the state-of-the-art in microarchitectural techniques for improving cache energy efficiency and motivate them to design novel solutions for addressing the challenges posed by future trends of CMOS fabrication and processor design.

References

[1] S. Murugesan, "Harnessing green IT: Principles and practices," *IT professional*, vol. 10, no. 1, pp. 24–33, 2008.

[2] S. Borkar, "Design challenges of technology scaling," *Micro, IEEE*, vol. 19, no. 4, pp. 23 –29, jul. 1999.

[3] G. Gammie, A. Wang, H. Mair, R. Lagerquist, M. Chau, P. Royannez, S. Gururajarao, and U. Ko, "Smartreflex power and performance management technologies for 90 nm, 65 nm, and 45 nm mobile application processors," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 144–159, 2010.

[4] "International technology roadmap for semiconductors (ITRS)," http://www.itrs.net/Links/2011ITRS/2011Chapters/2011ExecSum.pdf, 2011.

[5] S. Borkar, "Thousand core chips: a technology perspective," in *44th annual Design Automation Conference*. ACM, 2007, pp. 746–749.

[6] "First the Tick, Now the Tock: Next Generation Intel Microarchitecture (Nehalem)," Intel Whitepaper, Tech. Rep., 2008.

[7] B. Stackhouse *et al.*, "A 65 nm 2-billion transistor quad-core Itanium processor," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 18–31, 2009.

[8] R. Riedlinger, R. Bhatia, L. Biro, B. Bowhill, E. Fetzer, P. Gronowski, and T. Grutkowski, "A 32nm 3.1 billion transistor 12-wide-issue Itanium® processor for mission-critical servers," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2011, pp. 84–86.

[9] A. Vardhan and Y. Srikant, "Exploiting critical data regions to reduce data cache energy consumption," Indian Institute of Science, Bangalore, Tech. Rep., 2013.

[10] S. Li, J. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 469–480.

[11] B. Calder, D. Grunwald, and J. Emer, "Predictive sequential associative cache," in *International Symposium on High-Performance Computer Architecture*, 1996, pp. 244–253.

[12] K. Inoue, T. Ishihara, and K. Murakami, "Way-predicting set-associative cache for high performance and low energy consumption," in *international symposium on Low power electronics and design*, 1999, pp. 273–275.

[13] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *28th international symposium on Computer architecture (ISCA)*, 2001, pp. 240–251.

[14] M. Powell, A. Agrawal, T. Vijaykumar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," in *34th International Symposium on Microarchitecture*, 2001, pp. 54–65.

[15] P. Carazo Minguela, R. Apolloni, F. Castro, D. Chaver, L. Pinuel, and F. Tirado, "L1 Data Cache Power Reduction using a Forwarding Predictor," *Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation*, pp. 116–125, 2011.

[16] S. Kim and J. Lee, "Write buffer-oriented energy reduction in the L1 data cache of two-level caches for the embedded system," in *20th Great Lakes symposium on VLSI*. ACM, 2010, pp. 257–262.

[17] Z. Fang, L. Zhao, X. Jiang, S. Lu, R. Iyer, T. Li, and S. Lee, "Reducing L1 caches power by exploiting software semantics," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2012.

[18] D. Nicolaescu, B. Salamat, A. Veidenbaum, and M. Valero, "Fast speculative address generation and way caching for reducing L1 data cache energy," in *International Conference on Computer Design (ICCD)*, 2006, pp. 101–107.

[19] A. Bardine, M. Comparetti, P. Foglia, and C. A. Prete, "Evaluation of leakage reduction alternatives for deep submicron dynamic nonuniform cache architecture caches," in *IEEE Transactions on VLSI*, 2013.

[20] A. Bardine, M. Comparetti, P. Foglia, G. Gabrielli, C. Prete, and P. Stenström, "Leveraging data promotion for low power D-NUCA caches," in *11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD)*. IEEE, 2008, pp. 307–316.

[21] A. Udipi, N. Muralimanohar, and R. Balasubramonian, "Non-uniform power access in large caches with low-swing wires," in *International Conference on High Performance Computing (HiPC)*. IEEE, 2009, pp. 59–68.

[22] F. M. Sleiman, R. G. Dreslinski, and T. F. Wenisch, "Embedded way prediction for last-level caches," in *IEEE 30th International Conference on Computer Design (ICCD)*, 2012, pp. 167–174.

[23] S. Mittal and Z. Zhang, "EnCache: Improving cache energy efficiency using a software-controlled profiling cache," in *IEEE International Conference On Electro/Information Technology*, Indianapolis, USA, May 2012.

[24] S. Mittal, Z. Zhang, and Y. Cao, "CASHIER: A Cache Energy Saving Technique for QoS Systems," in *26th International Conference on VLSI Design*. IEEE, 2013, pp. 43–48.

[25] S. Mittal and Z. Zhang, "Palette: A cache leakage energy saving technique for green computing," in *HPC: Transition Towards Exascale Processing*, ser. Advances in Parallel Computing, C. Catlett, W. Gentzsch, L. Grandinetti, G. Joubert, and J. Vazquez-Poletti, Eds. IOS Press, 2013.

[26] S. Mittal and Z. Zhang, "MANAGER: A Multicore Shared Cache Energy Saving Technique for QoS Systems," Iowa State University, Tech. Rep., 2013.

[27] S. Mittal, "Dynamic cache reconfiguration based techniques for improving cache energy efficiency," Ph.D. dissertation, Iowa State University, 2013.

[28] R. Min, W. Jone, and Y. Hu, "Location cache: a low-power L2 cache system," in *International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2004, pp. 120–125.

[29] J. Abella, A. González, X. Vera, and M. O'Boyle, "IATAC: a smart predictor to turn-off L2 cache lines," *ACM Transactions on Architecture and Code Optimization*, vol. 2, no. 1, pp. 55–77, 2005.

[30] L. Li, I. Kadayif, Y. Tsai, N. Vijaykrishnan, M. Kandemir, M. Irwin, and A. Sivasubramaniam, "Leakage energy management in cache hierarchies," in *IEEE PACT*, 2002, pp. 131–140.

[31] S. Chung and K. Skadron, "On-demand solution to minimize I-Cache leakage energy with maintaining performance," *IEEE Transactions on Computers*, vol. 57, no. 1, pp. 7–24, 2008.

[32] N. Kim, K. Flautner, D. Blaauw, and T. Mudge, "Drowsy instruction caches. leakage power reduction using dynamic voltage scaling and cache sub-bank prediction," in *35th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2002, pp. 219–230.

[33] N. Kim, K. Flautner, D. Blaauw, and T. Mudge, "Single-V_{DD} and single-V_T super-drowsy techniques for low-leakage high-performance instruction caches," in *International Symposium on Low power electronics and design (ISLPED)*, 2004, pp. 54–57.

[34] J. Gu, H. Guo, and P. Li, "Robtic: An on-chip instruction cache design for low power embedded systems," in *15th IEEE RTCSA*, 2009, pp. 419–424.

[35] Z. Xie, D. Tong, and X. Cheng, "WHOLE: A low energy I-Cache with separate way history," in *IEEE International Conference on Computer Design*, 2009, pp. 137–143.

[36] Z. Ge, T. Mitra, and W. Wong, "A DVS-based pipelined reconfigurable instruction memory," in *Design Automation Conference*, 2009, pp. 897–902.

[37] Z. Hongwei, Z. Chengyi, and Z. Mingxuan, "Improved way prediction policy for low-energy instruction caches," *Embedded Software and Systems*, pp. 425–436, 2007.

[38] A. Gordon-Ross, S. Cotterell, and F. Vahid, "Tiny instruction caches for low power embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 2, no. 4, pp. 449–481, Nov. 2003.

[39] P. Kalla, X. S. Hu, and J. Henkel, "Distance-based recent use (DRU): an enhancement to instruction cache replacement policies for transition energy reduction," *IEEE Trans. on VLSI Syst.*, vol. 14, no. 1, pp. 69–80, 2006.

[40] T. Jones, S. Bartolini, B. De Bus, J. Cavazos, and F. O'Boyle, "Instruction cache energy saving through compiler way-placement," in *Design, Automation and Test in Europe (DATE)*. IEEE, 2008, pp. 1196–1201.

[41] W. Zhang, J. Hu, V. Degalahal, M. Kandemir, N. Vijaykrishnan, and M. Irwin, "Compiler-directed instruction cache leakage optimization," in *International Symposium on Microarchitecture (MICRO)*, 2002, pp. 208–218.

[42] J. Hu, A. Nadgir, N. Vijaykrishnan, M. Irwin, and M. Kandemir, "Exploiting program hotspots and code sequentiality for instruction cache leakage management," in *international symposium on Low power electronics and design*. ACM, 2003, pp. 402–407.

[43] S.-H. Yang, B. Falsafi, M. D. Powell, K. Roy, and T. N. Vijaykumar, "An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-caches," in *7th International Symposium on High-Performance Computer Architecture (HPCA)*, 2001.

[44] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. Vijaykumar, "Gated-Vdd: a circuit technique to reduce leakage in deep-submicron cache memories," in *International Symposium on Low power electronics and design (ISLPED)*, 2000, pp. 90 – 95.

[45] W. Zhang, M. Karakoy, M. Kandemir, and G. Chen, "A compiler approach for reducing data cache energy," in *17th annual international conference on Supercomputing*. ACM, 2003, pp. 76–85.

[46] J. Yang and R. Gupta, "Energy efficient frequent value data cache design," in *International Symposium on Microarchitecture (MICRO)*, 2002, pp. 197–207.

[47] T. Kalyan and M. Mutyam, "Word-interleaved cache: an energy efficient data cache architecture," in *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2008, pp. 265–270.

[48] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power," in *International Symposium on Computer Architecture (ISCA)*, 2002, pp. 148–157.

[49] Z. Zhu and X. Zhang, "Access-mode predictions for low-power cache design," *Micro, IEEE*, vol. 22, no. 2, pp. 58–71, 2002.

[50] G. Keramidas, P. Xekalakis, and S. Kaxiras, "Applying decay to reduce dynamic power in set-associative caches," *High Performance Embedded Architectures and Compilers*, pp. 38–53, 2007.

[51] H. Hanson, M. Hrishikesh, V. Agarwal, S. Keckler, and D. Burger, "Static energy reduction techniques for microprocessor caches," *IEEE Transactions on VLSI Systems*, vol. 11, no. 3, pp. 303 –313, 2003.

[52] S. Petit, J. Sahuquillo, J. Such, and D. Kaeli, "Exploiting temporal locality in drowsy cache policies," in *2nd conference on Computing frontiers*. ACM, 2005, pp. 371–377.

[53] N. Mohyuddin, R. Bhatti, and M. Dubois, "Controlling leakage power with the replacement policy in slumberous caches," in *2nd conference on Computing frontiers*. ACM, 2005, pp. 161–170.

[54] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M. L. Scott, "Integrating adaptive on-chip storage structures for reduced dynamic power," in *PACT*, 2002.

[55] M. Rawlins and A. Gordon-Ross, "On the interplay of loop caching, code compression, and cache configuration," in *16th Asia and South Pacific Design Automation Conference*, 2011, pp. 243–248.

[56] I. Kotera, K. Abe, R. Egawa, H. Takizawa, and H. Kobayashi, "Power-aware dynamic cache partitioning for CMPs," *Transactions on high-performance embedded architectures and compilers III*, pp. 135–153, 2011.

[57] C. Yu and P. Petrov, "Aggressive snoop reduction for synchronized producer-consumer communication in energy-efficient embedded multi-processors," in *5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, 2007, pp. 245–250.

[58] Y. Guo, P. Narayanan, M. A. Bennaser, S. Chheda, and C. A. Moritz, "Energy-Efficient Hardware Data Prefetching," *IEEE Trans. VLSI Syst.*, vol. 19, no. 2, p. 250263, 2011.

[59] S. Rodriguez and B. Jacob, "Energy/power breakdown of pipelined nanometer caches (90nm/65nm/45nm/32nm)," in *international symposium on Low power electronics and design*. ACM, 2006, pp. 25–30.

[60] J. Butts and G. Sohi, "A static power model for architects," in *international symposium on Microarchitecture*, 2000, pp. 191–201.

[61] D. Helms, E. Schmidt, and W. Nebel, "Leakage in CMOS circuits–an introduction," *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, pp. 17–35, 2004.

[62] S. Kaxiras and M. Martonosi, "Computer architecture techniques for power-efficiency," *Synthesis Lectures on Computer Architecture*, vol. 3, no. 1, pp. 1–207, 2008.

[63] J. Kin, M. Gupta, and W. Mangione-Smith, "The filter cache: an energy efficient memory structure," in *30th International symposium on Microarchitecture (MICRO)*, 1997, pp. 184–193.

[64] Y.-Y. Tsai and C.-H. Chen, "Energy-Efficient Trace Reuse Cache for Embedded Processors." *IEEE Trans. VLSI Syst.*, vol. 19, no. 9, p. 16811694, 2011.

[65] R. Giorgi and P. Bennati, "Reducing leakage in power-saving capable caches for embedded systems by using a filter cache," in *workshop on MEmory performance: DEaling with Applications, systems and architecture*, 2007, pp. 97–104.

[66] C. Ballapuram, A. Sharif, and H. Lee, "Exploiting access semantics and program behavior to reduce snoop power in chip multiprocessors," in *ACM Sigplan Notices*, vol. 43, no. 3, 2008, pp. 60–69.

[67] M. Ghosh, E. Ozer, S. Ford, S. Biles, and H. Lee, "Way guard: a segmented counting bloom filter approach to reducing energy for set-associative caches," in *International Symposium on Low Power Electronics and Design*, 2009, pp. 165–170.

[68] G. Memik, G. Reinman, and W. Mangione-Smith, "Just say no: Benefits of early cache miss determination," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2003, pp. 307–316.

[69] Z. Mingming, C. Xiaotao, and Z. Ge, "Reducing cache energy consumption by tag encoding in embedded processors," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2007, pp. 367–370.

[70] K. T. Sundararajan, V. Porpodas, T. M. Jones, N. P. Topham, and B. Franke, "Cooperative partitioning: Energy-efficient cache partitioning for high-performance CMPs," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2012.

[71] H. Park, S. Yoo, and S. Lee, "A multistep tag comparison method for a low-power L2 cache," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 4, pp. 559–572, 2012.

[72] J. Kwak and Y. Jeon, "Compressed tag architecture for low-power embedded cache systems," *Journal of Systems Architecture*, vol. 56, no. 9, pp. 419–428, 2010.

[73] M. Loghi, P. Azzoni, and M. Poncino, "Tag overflow buffering: Reducing total memory energy by reduced-tag matching," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 5, pp. 728 –732, may 2009.

[74] A. Shafiee, N. Shahidi, and A. Baniasadi, "Using partial tag comparison in low-power snoop-based chip multiprocessors," in *Computer Architecture*, ser. Lecture Notes in Computer Science, A. Varbanescu, A. Molnos, and R. van Nieuwpoort, Eds. Springer, 2012, vol. 6161, pp. 211–221.

[75] J. Nemeth, R. Min, W. Jone, and Y. Hu, "Location cache design and performance analysis for chip multiprocessors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 1, pp. 104–117, 2011.

[76] Y. K. Cho, S. T. Jhang, and C. S. Jhon, "Selective word reading for high performance and low power processor," in *ACM Symposium on Research in Applied Computation*, 2011, pp. 25–30.

[77] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking (TON)*, vol. 8, no. 3, pp. 281–293, 2000.

[78] A. Agarwal, H. Li, and K. Roy, "DRG-cache: a data retention gated-ground cache for low power," in *Design Automation Conference*, 2002, pp. 473–478.

[79] Y. Zhao, X. Li, D. Tong, and X. Cheng, "Reuse distance based cache leakage control," in *14th international conference on High performance computing*. Springer-Verlag, 2007, pp. 356–367.

[80] H. Zhou, M. Toburen, E. Rotenberg, and T. Conte, "Adaptive mode control: A static-power-efficient cache design," *ACM Transactions on Embedded Computing Systems*, vol. 2, no. 3, pp. 347–372, 2003.

[81] K. Sundararajan, T. Jones, and N. Topham, "Smart cache: A self adaptive cache architecture for energy efficiency," in *International Conference on Embedded Computer Systems (SAMOS)*. IEEE, 2011, pp. 41–50.

[82] D. Benitez, J. Moure, D. Rexachs, and E. Luque, "A reconfigurable cache memory with heterogeneous banks," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*. IEEE, 2010, pp. 825–830.

[83] K. Tanaka and T. Kawahara, "Leakage energy reduction in cache memory by data compression," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 5, pp. 17–24, 2007.

[84] J. Li and Y. Hwang, "Snug set-associative caches: reducing leakage power while improving performance," in *international symposium on Low power electronics and design*, 2005, pp. 345–350.

[85] H. Kim and J. Kim, "A leakage-aware L2 cache management technique for producer–consumer sharing in low-power chip multiprocessors," in *Journal of Parallel and Distributed Computing*. Elsevier, 2011.

[86] M. A. Z. Alves *et al.*, "Energy savings via dead sub-block prediction," in *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2012.

[87] M. Ghosh and H. Lee, "Virtual exclusion: An architectural approach to reducing leakage energy in caches for multiprocessor systems," in *International Conference on Parallel and Distributed Systems*, vol. 2. IEEE, 2007, pp. 1–8.

[88] S. Kaxiras, P. Xekalakis, and G. Keramidas, "A simple mechanism to adapt leakage-control policies to temperature," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2005, pp. 54–59.

[89] D. H. Albonesi, "Selective cache ways: on-demand cache resource allocation," in *International Symposium on Microarchitecture*, 1999, pp. 248–259.

[90] K. Meng, R. Joseph, R. Dick, and L. Shang, "Multi-optimization power management for chip multiprocessors," in *PACT*, 2008, pp. 177–186.

[91] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache architecture for embedded systems," in *international symposium on Computer architecture (ISCA)*, 2003, pp. 136–146.

[92] J. Ku, S. Ozdemir, G. Memik, and Y. Ismail, "Thermal management of on-chip caches through power density minimization," in *International Symposium on Microarchitecture (MICRO)*, 2005, pp. 283–293.

[93] H. Noori, M. Goudarzi, K. Inoue, and K. Murakami, "Improving energy efficiency of configurable caches via temperature-aware configuration selection," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2008, pp. 363–368.

[94] S. Yang, B. Falsafi, M. Powell, and T. Vijaykumar, "Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2002, pp. 151–161.

[95] G. Keramidas, C. Datsios, and S. Kaxiras, "A framework for efficient cache resizing," in *International Conference on Embedded Computer Systems (SAMOS)*. IEEE, 2012, pp. 76–85.

[96] H. Kim, J. Ahn, and J. Kim, "Replication-aware leakage management in chip multiprocessors with private L2 cache," in *international symposium on Low power electronics and design*, 2010, pp. 135–140.

[97] R. Reddy and P. Petrov, "Cache partitioning for energy-efficient and interference-free embedded multitasking," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 9, no. 3, p. 16, 2010.

[98] W. Wang and P. Mishra, "Dynamic reconfiguration of two-level caches in soft real-time embedded systems," in *IEEE Computer Society Annual Symposium on VLSI*, 2009, pp. 145–150.

[99] W. Wang, P. Mishra, and S. Ranka, "Dynamic cache reconfiguration and partitioning for energy optimization in real-time multicore systems," in *48th Design Automation Conference*, 2011, pp. 948–953.

[100] K. T. Sundararajan, T. M. Jones, and N. P. Topham, "The smart cache: An energy-efficient cache architecture through dynamic adaptation," in *International Journal of Parallel Programming*, 2012.

[101] L. Yuan, S. Leventhal, J. Gu, and G. Qu, "TALk: A Temperature-Aware Leakage Minimization Technique for Real-Time Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 10, pp. 1564–1568, 2011.

[102] L. He, W. Liao, and M. Stan, "System level leakage reduction considering the interdependence of temperature and leakage," in *Design Automation Conference*, 2004, pp. 12–17.

[103] M. Sato, R. Egawa, H. Takizawa, and H. Kobayashi, "A voting-based working set assessment scheme for dynamic cache resizing mechanisms," in *IEEE International Conference on Computer Design (ICCD)*, 2010, pp. 98–105.

[104] K. Kedzierski, F. Cazorla, R. Gioiosa, A. Buyuktosunoglu, and M. Valero, "Power and performance aware reconfigurable cache for CMPs," in *Second International Forum on Next-Generation Multicore/Manycore Technologies*. ACM, 2010.

[105] X. Jiang, A. Mishra, L. Zhao, R. Iyer, Z. Fang, S. Srinivasan, S. Makineni, P. Brett, and C. Das, "ACCESS: Smart scheduling for asymmetric cache CMPs," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2011, pp. 527–538.

[106] M. Rawlins and A. Gordon-Ross, "CPACT-The conditional parameter adjustment cache tuner for dual-core architectures," in *International Conference on Computer Design (ICCD)*, 2011, pp. 396–403.

[107] M. Monchiero, R. Canal, and A. Gonzalez, "Using coherence information and decay techniques to optimize L2 cache leakage in CMPs," in *International Conference on Parallel Processing (ICPP)*, 2009, pp. 1–8.

[108] J. Zhao, C. Xu, and Y. Xie, "Bandwidth-aware reconfigurable cache design with hybrid memory technologies," in *International Conference on Computer-Aided Design*. IEEE Press, 2010, pp. 48–55.

[109] H. Ghasemi, S. Draper, and N. S. Kim, "Low-voltage on-chip cache architecture using heterogeneous cell sizes for high-performance processors," in *International Symposium on High Performance Computer Architecture (HPCA)*, feb. 2011, pp. 38 –49.

[110] K. Kedzierski, M. Moreto, F. Cazorla, and M. Valero, "Adapting cache partitioning algorithms to pseudo-LRU replacement policies," in *International Symposium on Parallel & Distributed Processing (IPDPS)*, 2010, pp. 1–12.

[111] X. Fu, K. Kabir, and X. Wang, "Cache-aware utilization control for energy efficiency in multi-core real-time systems," in *23rd Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 2011, pp. 102–111.

[112] H. Hajimiri, P. Mishra, and S. Bhunia, "Dynamic cache tuning for efficient memory based computing in multicore architectures," in *IEEE International Conference on VLSI Design*, 2013.

[113] M. Lodde *et al.*, "Dynamic last-level cache allocation to reduce area and power overhead in directory coherence protocols," in *Euro-Par 2012 Parallel Processing*, ser. Lecture Notes in Computer Science, C. Kaklamanis, T. Papatheodorou, and P. Spirakis, Eds. Springer, 2012, vol. 7484, pp. 206–218.

[114] A. Nacul and T. Givargis, "Dynamic voltage and cache reconfiguration for low power," in *Design, automation and test in Europe-Volume 2*. IEEE Computer Society, 2004.

[115] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Design Automation Conference*. IEEE, 2004, pp. 275–280.

[116] W. Wang and P. Mishra, "Leakage-aware energy minimization using dynamic voltage scaling and cache reconfiguration in real-time systems," in *23rd International Conference on VLSI Design*, 2010, pp. 357–362.

[117] H. Hajimiri, K. Rahmani, and P. Mishra, "Synergistic integration of dynamic cache reconfiguration and code compression in embedded systems," in *International Green Computing Conference and Workshops (IGCC)*. IEEE, 2011, pp. 1–8.

[118] S. Kim, J. Lee, J. Kim, and S. Hong, "Residue cache: a low-energy low-area L2 cache architecture via compression and partial hits," in *International Symposium on Microarchitecture*, 2011, pp. 420–429.

[119] I. Kadayif, A. Zorlubas, S. Koyuncu, O. Kabal, D. Akcicek, Y. Sahin, and M. Kandemir, "Capturing and optimizing the interactions between prefetching and cache line turnoff," *Microprocessors and Microsystems*, vol. 32, no. 7, pp. 394–404, 2008.

[120] Y.-J. Chen, C.-L. Yang, J.-W. Chi, and J.-J. Chen, "TACLC: Timing-Aware Cache Leakage Control for Hard Real-Time Systems." *IEEE Trans. Computers*, vol. 60, no. 6, p. 767782, 2011.

[121] W. Wang, S. Ranka, and P. Mishra, "A general algorithm for energy-aware dynamic reconfiguration in multitasking systems," in *24th International Conference on VLSI Design*, 2011, pp. 334–339.

[122] M. Paul and P. Petrov, "Dynamically Adaptive I-Cache Partitioning for Energy-Efficient Embedded Multitasking." *IEEE Trans. VLSI Syst.*, vol. 19, no. 11, p. 20672080, 2011.

[123] G. H. Loh, "3D-stacked memory architectures for multi-core processors," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, 2008, pp. 453–464.

[124] http://www.tezzaron.com/technology/FaStack.htm, 2013.

[125] B. Black *et al.*, "Die stacking (3D) microarchitecture," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2006, pp. 469–479.

[126] D. Jevdjic, S. Volos, and B. Falsafi, "Die-Stacked DRAM Caches for Servers: Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache," in *International Symposium on Computer architecture (ISCA)*, 2013.

[127] H. Sun, J. Liu, R. S. Anigundi, N. Zheng, J.-Q. Lu, K. Rose, and T. Zhang, "3D DRAM design and application to 3D multicore systems," *IEEE Design & Test of Computers*, vol. 26, no. 5, pp. 36–47, 2009.

[128] Y. Li, D. Parikh, Y. Zhang, K. Sankaranarayanan, M. Stan, and K. Skadron, "State-preserving vs. non-state-preserving leakage control in caches," in *Design, Automation and Test in Europe Conference and Exhibition*, vol. 1. IEEE, 2004, pp. 22–27.

[129] K. Nii, H. Makino, Y. Tujihashi, C. Morishima, Y. Hayakawa, H. Nunogami, T. Arakawa, and H. Hamano, "A low power SRAM using auto-backgate-controlled MT-CMOS," in *International Symposium on Low Power Electronics and Design*. IEEE, 1998, pp. 293–298.

[130] R. L. Mattson, "Evaluation techniques in storage hierarchies," *IBM Journal of research and development*, vol. 9, 1970.

[131] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *International Symposium on Microarchitecture*, 2006, pp. 423–432.

[132] Y. Wang, S. Roy, and N. Ranganathan, "Run-time power-gating in caches of GPUs for leakage energy savings," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, march 2012, pp. 300 –303.

[133] J. Abella and A. González, "Heterogeneous way-size cache," in *international conference on Supercomputing*. ACM, 2006, pp. 239–248.

[134] W. Wong, C. Koh, Y. Chen, and H. Li, "VOSCH: Voltage scaled cache hierarchies," in *25th International Conference on Computer Design (ICCD)*. IEEE, 2007, pp. 496–503.

[135] A. Malik, B. Moyer, and D. Cermak, "A low power unified cache architecture providing power and performance flexibility," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2000, pp. 241–243.

[136] G. Gerosa, S. Curtis, M. D'Addeo, B. Jiang, B. Kuttanna, F. Merchant, B. Patel, M. Taufique, and H. Samarchi, "A sub-1W to 2W low-power IA processor for mobile internet devices and ultra-mobile PCs in 45nm hi-κ metal gate CMOS," in *IEEE International Solid-State Circuits Conference (ISSCC). Digest of Technical Papers.*, 2008, pp. 256–611.

[137] J. Chang, M. Huang, J. Shoemaker, J. Benoit, S. Chen, W. Chen, S. Chiu, R. Ganesan, G. Leong, V. Lukka *et al.*, "The 65-nm 16-MB Shared On-Die L3 Cache for the Dual-Core Intel Xeon Processor 7100 Series," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 4, pp. 846–852, 2007.

[138] S. Rusu, S. Tam, H. Muljono, J. Stinson, D. Ayers, J. Chang, R. Varada, M. Ratta, S. Kottapalli, and S. Vora, "A 45 nm 8-core enterprise Xeon processor," *IEEE Journal of Solid-State Circuits,*, vol. 45, no. 1, pp. 7–14, 2010.

[139] N. Kurd, S. Bhamidipati, C. Mozak, J. Miller, T. Wilson, M. Nemani, and M. Chowdhury, "Westmere: A family of 32nm IA processors," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2010, pp. 96–97.

[140] N. Sakran, M. Yuffe, M. Mehalel, J. Doweck, E. Knoll, and A. Kovacs, "The implementation of the 65nm dual-core 64b Merom processor," in *IEEE International Solid-State Circuits Conference (ISSCC). Digest of Technical Papers*, 2007, pp. 106–590.

[141] P. Royannez, H. Mair, F. Dahan, M. Wagner, M. Streeter, L. Bouetel, J. Blasquez, H. Clasen, G. Semino, J. Dong *et al.*, "90nm low leakage SoC design techniques for wireless applications," in *IEEE International Solid-State Circuits Conference (ISSCC), Digest of Technical Papers*, 2005, pp. 138–589.

[142] H. Mair, A. Wang, G. Gammie, D. Scott, P. Royannez, S. Gururajarao, M. Chau, R. Lagerquist, L. Ho, M. Basude *et al.*, "A 65-nm mobile multimedia applications processor with an adaptive power management scheme to compensate for variations," in *IEEE Symposium on VLSI Circuits*, 2007, pp. 224–225.

[143] G. Gammie, A. Wang, M. Chau, S. Gururajarao, R. Pitts, F. Jumel, S. Engel, P. Royannez, R. Lagerquist, H. Mair *et al.*, "A 45nm 3.5 g baseband-and-multimedia application processor using adaptive body-bias and ultra-low-power techniques," in *IEEE International Solid-State Circuits Conference (ISSCC), Digest of Technical Papers*, 2008, pp. 258–611.

[144] V. George, S. Jahagirdar, C. Tong, K. Smits, S. Damaraju, S. Siers, V. Naydenov, T. Khondker, S. Sarkar, and P. Singh, "Penryn: 45-nm next generation Intel® core 2 processor," in *IEEE Asian Solid-State Circuits Conference (ASSCC)*, 2007, pp. 14–17.

[145] A. Branover, D. Foley, and M. Steinman, "AMD Fusion APU: Llano." *IEEE Micro*, vol. 32, no. 2, p. 2837, 2012.

[146] K. Zhang, U. Bhattacharya, Z. Chen, F. Hamzaoglu, D. Murray, N. Vallepalli, Y. Wang, B. Zheng, and M. Bohr, "SRAM design on 65-nm CMOS technology with dynamic sleep transistor for leakage reduction," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 4, pp. 895–901, 2005.

[147] Y. Wang, U. Bhattacharya, F. Hamzaoglu, P. Kolar, Y. Ng, L. Wei, Y. Zhang, K. Zhang, and M. Bohr, "A 4.0 GHz 291 Mb voltage-scalable SRAM design in a 32 nm high-k+ metal-gate CMOS technology with integrated power management," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 1, pp. 103–110, 2010.

[148] F. Hamzaoglu, K. Zhang, Y. Wang, H. Ahn, U. Bhattacharya, Z. Chen, Y. Ng, A. Pavlov, K. Smits, and M. Bohr, "A 3.8 GHz 153 Mb SRAM design with dynamic stability enhancement and leakage reduction in 45 nm high-k metal gate CMOS technology," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 148–154, 2009.

[149] Y. Wang, H. Ahn, U. Bhattacharya, T. Coan, F. Hamzaoglu, W. Hafez, C. Jan, R. Kolar, S. Kulkarni, J. Lin *et al.*, "A 1.1 GHz 12μA/Mb-Leakage SRAM Design in 65nm Ultra-Low-Power CMOS with Integrated Leakage Reduction For Mobile Applications," in *IEEE International Solid-State Circuits Conference (ISSCC), Digest of Technical Papers.*, 2007, pp. 324–606.

[150] M. Khellah, N. Kim, J. Howard, G. Ruhl, Y. Ye, J. Tschanz, D. Somasekhar, N. Borkar, F. Hamzaoglu, G. Pandya *et al.*, "A 4.2 GHz 0.3 mm^2 256kb Dual-V/sub cc/SRAM Building Block in 65nm CMOS," in *IEEE International Solid-State Circuits Conference (ISSCC), Digest of Technical Papers.*, 2006, pp. 2572–2581.