FlexiWay: A Cache Energy Saving Technique Using Fine-grained Cache Reconfiguration

Sparsh Mittal, Zhao Zhang, Jeffrey S. Vetter

Published in IEEE ICCD, 2013. DOI:10.1109/ICCD.2013.6657031

FlexiWay: A Cache Energy Saving Technique Using Fine-grained Cache Reconfiguration

Sparsh Mittal, Zhao Zhang
Electrical and Computer Engineering Department
Iowa State University, USA
sparsh0mittal@gmail.com, zzhang@iastate.edu

Jeffrey S. Vetter
Future Technologies Group
Oak Ridge National Laboratory, USA
vetter@computer.org

***Abstract*—Recent trends of CMOS scaling and use of large last level caches (LLCs) have led to significant increase in the leakage energy consumption of LLCs and hence, managing their energy consumption has become extremely important in modern processor design. The conventional cache energy saving techniques require offline profiling or provide only coarse granularity of cache allocation. We present FlexiWay, a cache energy saving technique which uses dynamic cache reconfiguration. FlexiWay logically divides the cache sets into multiple (e.g. 16) *modules* and dynamically turns off suitable and *possibly different* number of cache ways in each module. FlexiWay has very small implementation overhead and it provides fine-grain cache allocation even with caches of typical associativity, e.g. an 8-way cache. Microarchitectural simulations have been performed using an x86-64 simulator and workloads from SPEC2006 suite. Also, FlexiWay has been compared with two conventional energy saving techniques. The results show that FlexiWay provides largest energy saving and incurs only small loss in performance. For single, dual and quad core systems, the average energy saving using FlexiWay are 26.2%, 25.7% and 22.4%, respectively.**

***Index Terms*—Cache leakage energy saving, way-based cache reconfiguration, energy efficiency, low-power, green computing.**

I. Introduction

A large value of power consumption of modern processors has been identified as the most severe obstacle in scaling their performance [1]. Further, due to several recent trends, the energy consumption of large last-level caches (LLCs) is becoming a significant fraction of processor energy consumption. Since LLC is the last line of defense before hitting the memory wall, modern processors are using increasingly large LLCs to bridge the gap between the speed of processor and main memory, for example, Intel's 32nm Sandy Bridge Core i7-3960X processor uses 15MB LLC. Figure 1 shows the die photo of this processor [2]. Clearly, LLCs consume a large fraction of the die area. Further, with recent CMOS technology generations, the leakage energy consumption has been dramatically increasing [3] and hence, leakage energy consumption of large LLCs is also on rise. The increased levels of power consumption also increase the cost of cooling and lead to increased complexity in chip packaging. For these reasons, managing the power consumption of LLCs has become extremely important in modern processor design.

Conventional cache energy saving techniques have several limitations. Some techniques use offline profiling to find the value of its parameters for each application [4]–[7]. However,

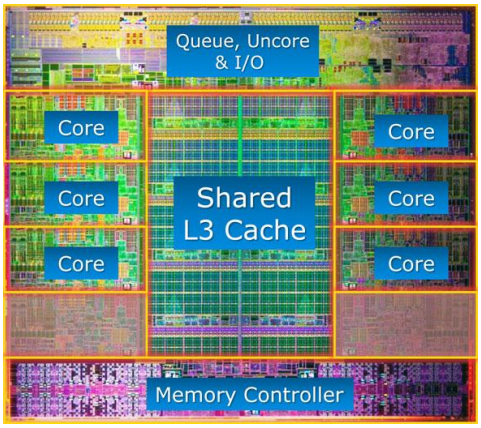

Fig. 1. Die Photo of Intel's 32nm Sandy Bridge Core i7-3960X [2]. Notice that LLC consumes a significant amount of chip area.

in multicore processors the possible combinations of applications increase greatly and hence, offline profiling becomes difficult. The selective-way based techniques [6]–[10] provide only few coarse grain partitions (at most, as many as the number of ways). The selective-sets [5], hybrid (selective-sets and selective-ways) [11] and cache-coloring [12] based techniques alter set-decoding of the cache and hence, these techniques incur large flushing overhead on reconfiguration.

In this paper, we present FlexiWay, a dynamic cache reconfiguration based approach for saving leakage energy. FlexiWay works on the following observation. It is well-known that accesses to the cache sets are non-uniformly distributed; thus, some sets see many more cache accesses than the other sets. Further, the associativity requirement of different sets is also different. To illustrate this, we assume that the LLC is an L2 cache and logically divide the L2 sets into multiple (e.g. 8) groups called *modules* and term the ways of a module as *sub-ways*. Further, in Figure 2, we show the number of hits to different sub-ways of the L2 cache for h264ref benchmark. From the figure, it is clear that the modules marked in light gray (viz. 2,3,4,5) require larger associativity than those marked in dark gray (viz. 0,1,6,7). For such cases, conventional selective-ways technique turn-off exactly same number of ways for all the modules (or sets). In contrast, FlexiWay works by turning-off suitable and *possibly different* number of cache ways in each module. This provides fine-grain cache reconfiguration with caches of typical associativity (e.g. 8 way cache). Thus,

FlexiWay avoids the need of using caches of large associativity for achieving fine-grain reconfiguration.

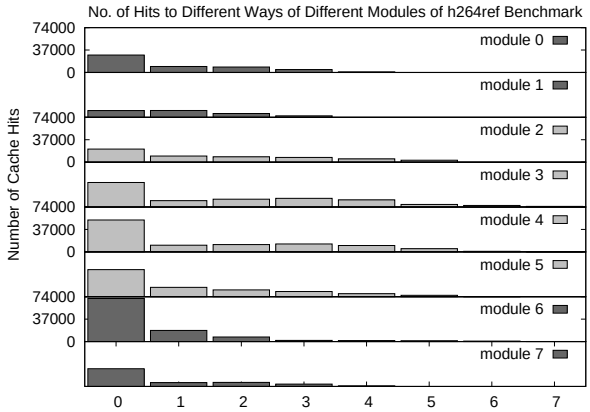

Fig. 2. Number of hits to different ways of different modules for h264ref benchmark. X-axis shows way number in LRU ordering chain in decreasing order of recency. L2 size is 4MB, execution length is 200M instructions and remaining parameters are shown in Section VII. For all sub-plots, Y-axis ranges from 0 to 74,000. Notice that modules marked in light gray require larger associativity than those marked in dark gray.

We evaluate FlexiWay using out-of-order simulations with Sniper x86-64 simulator and multi-programmed workloads from SPEC2006 suite (Section VII). We compare it to way adaptable cache technique (WAC) [10] which uses selective-ways approach and decay cache technique (DECAY) [13] (see Section VII-D for details). The results show that FlexiWay saves largest amount of memory subsystem (LLC+DRAM) energy (Section VIII). Over a shared baseline LLC, for single, dual and quad-core systems, the average savings in memory subsystem energy by using FlexiWay are 26.2%, 25.7% and 22.4%, respectively. Using WAC, these values are only 13.4%, 13.4% and 10.6%, respectively and using DECAY, these values are 23.5%, 20.2% and 16.6%, respectively. Further, FlexiWay incurs only small loss in performance and does not cause unfairness. Additional experimental results confirm that FlexiWay works well for a wide variety of system parameters.

II. Background and Related Work

Several previous works report the phenomenon of unbalanced accesses to different sets of the cache. Rolán et al. [14] propose a technique to achieve balanced accesses to different sets by extending the space available with underutilized sets to the heavily utilized sets. Their technique aims to improve performance and does not turn-off cache blocks. Moreover, it achieves balancing at the level of individual set. In contrast, FlexiWay improves energy efficiency by cache reconfiguration while minimizing performance loss and works at the level of a module, which has much larger number of sets (e.g. 512 sets).

V-way cache technique [15] increases the number of tag-store entries relative to the number of data lines and uses this to change the associativity on a per-set basis. It breaks the static one-to-one mapping between tags and data, and thus increases decoding overhead and also necessitates serial lookup. Heterogeneous way-size cache [16] uses different number of sets in each cache way, and adapts the size of each way according to the program requirement. Amorphous cache [17] uses heterogeneous sub-caches, which allow adapting the total cache size and/or set-associativity to the program requirement. However, these techniques require significant changes to the cache geometry and design. FlexiWay does not require such changes, instead, it uses cache reconfiguration to dynamically adapt the associativity for different modules.

Drowsy cache technique [18] uses state-preserving leakage control for saving cache energy. The limitation of this technique is that it requires two voltage supplies which increases the design complexity [18]. Also, it increases single-bit and multiple-bit errors [19] which necessitate more complex error detection/correction codes. Unlike a few techniques (e.g. [7]), which use static cache reconfiguration, FlexiWay uses dynamic cache reconfiguration which is important for realizing large energy savings, since the applications show a significant variation in their behavior over their execution length. Several previous cache energy saving techniques (e.g. [8], [20]) have been evaluated without considering their impact on components, other than cache. We include both LLC and DRAM energy for a more comprehensive evaluation.

III. Energy Model

We now discuss our energy model and use the terms introduced here in showing the working of FlexiWay in the next section. In this paper, the LLC is an L2 cache, and based on this FlexiWay can be easily shown to work for the case when the LLC is an L3 cache. We model the energy spent in L2 cache, DRAM and the overhead of algorithm (viz. energy cost of block-transitions). Our notation is shown in Table I.

TABLE I
The notation used in the paper

N	Number of cores
E_{xyz}^{Dyn}	Dynamic energy *per access* in xyz (L2 or DRAM)
P_{xyz}^{Leak}	Leakage energy *per second* in xyz (L2 or DRAM)
B	Number of cache block transitions
E_{χ}, E_{tran}	Energy consumed in single and all block transitions
H_{L2}, M_{L2}	Number of L2 hits and misses
F_A	Active fraction of cache
T	Time length of an interval (in seconds)
W	L2 associativity
w	Number of ways consulted in each cache access
A_{DRAM}	Number of DRAM accesses
P_{off}	Leakage power consumption in low-power mode as a fraction of normal leakage power, P_{L2}^{Leak}
Υ	Area overhead of gated V_{dd} memory cell as a fraction of area of the normal memory cell.

FlexiWay and DECAY need to consult all the ways in an access, and hence, for them, $w = W$. For WAC, w equals the number of active cache ways, since it turns off equal number of ways in all the sets. To compute L2 leakage energy, we account for the power consumption of both active and low-leakage portion of the cache. For computing L2 dynamic energy, an L2 miss is assumed to consume twice the dynamic energy as that of an L2 hit [11], [21]. The dynamic energy consumed in each access scales with w [7], [9]. Thus,

$$E = E_{L2} + E_{DRAM} + E_{Algo} \quad (1)$$

$$E_{L2} = LE_{L2} + DE_{L2} \quad (2)$$

$$LE_{L2} = P_{L2}^{Leak}(1 + \Upsilon)(F_A + (1 - F_A)P_{off}) \times T \quad (3)$$

$$DE_{L2} = E_{L2}^{Dyn} \times (2M_{L2} + H_{L2}) \times (\frac{w}{W}) \quad (4)$$

$$E_{DRAM} = P_{DRAM}^{Leak} \times T + E_{DRAM}^{Dyn} \times A_{DRAM} \quad (5)$$

$$E_{Algo} = E_{\chi} \times B \quad (6)$$

For baseline experiments, $E_{Algo} = 0$, F_A = 1, Υ = 0 and P_{off} value is not required. All the three techniques, viz. FlexiWay, WAC and DECAY use gated V_{dd} scheme [10], [13], for which Υ = 0.05 and P_{off} = 0.03 [22]. The values of E_{L2}^{Dyn} and P_{L2}^{Leak} are obtained using CACTI [23] assuming 8-bank structure at 45nm. These values are shown in Table II in section VII. E_{DRAM}^{Dyn} and P_{DRAM}^{Leak} are taken as 70 nJ and 0.18 Watt, respectively [11], [24] and E_{χ} is taken as 2 pJ [11].

IV. FlexiWay: System Architecture

In this section, we first discuss the intuition behind working of FlexiWay. Then, we discuss the method for dynamically collecting profiling information. Finally, we discuss the criterion for turning-off a sub-way.

A. Motivation and Main Idea

In contrast with two extremes, viz. fully set-associative cache and direct-mapped cache, caches of typical set-associativity (e.g. 4, 8 or 16 ways) provide a balance between the goals of avoiding the conflict misses and reducing cache access time. However, for several applications, different sets show different set-associative requirements. The conventional selective-ways based techniques turn-off exactly same number of ways in all the sets of the cache. Because of this, these techniques cannot achieve fine-grained cache reconfiguration, since their reconfiguration granularity is limited by the number of ways in the cache. Thus, to achieve fine-grained reconfiguration, these techniques must use caches of higher associativity which have significantly high access time and energy.

Clearly, turning off different number of cache ways in each set can provide larger energy savings. Since collecting profiling information for each set and reconfiguring on the granularity of single set would incur prohibitive overhead, we instead divide the sets into Q modules and change associativity at the granularity of each module. Then, FlexiWay observes the number of hits to different ways of all the modules and uses this information to dynamically turn-off possibly different number of ways in each module. Typical values of Q can be 8, 16, 32 etc. FlexiWay keeps at least W_{min} ($\geq$ 2) ways always on, since on keeping only one way turned-on, we get a direct-mapped LLC, which may lead to severe cache conflicts. By changing W_{min}, a balance between energy saving and performance loss can be achieved. Thus, FlexiWay provides $Q \times (W - W_{min})$ levels of cache allocation granularity.

B. Collecting Profiling Information

To take reconfiguration decisions, FlexiWay uses dynamic profiling. For this purpose, an auxiliary tag directory (ATD) is used which uses set-sampling with a sampling-ratio (R_S) of 64. Thus, it monitors only 1/64 fraction of the cache sets.

There are two possible methods for using ATD. **First**, embedding the functionality of ATD in the main tag directory (MTD) of L2 cache itself. Profiling information is collected from only few sets, called leader sets and they do not undergo cache turnoff. The remaining sets, called follower sets, undergo cache turnoff based on the decision of the algorithm. This is similar to previous work [25]. **Second**, using a separate ATD. Since this ATD does not store data and uses large sampling ratio, its overhead is small. For a tag-size of 30 bits, the overhead of ATD is only 0.08% of the L2 cache. Unless otherwise stated, in this paper we use the first method. In Section VIII-B, we evaluate FlexiWay with the second method. The results have shown that the both methods provide nearly equal energy saving and performance.

The hits to a particular set in ATD count towards the module in which that set is present. For each module, W counters are used and a hit to a way in a set of ATD increments the counter of that way in the corresponding module. For illustration, we take a hypothetical case where a cache has 64 sets and Q = 4 and R_S = 8. Then, as shown in Figure 3, the ATD has 8 sets, and 2 ATD (leader) sets correspond to each module.

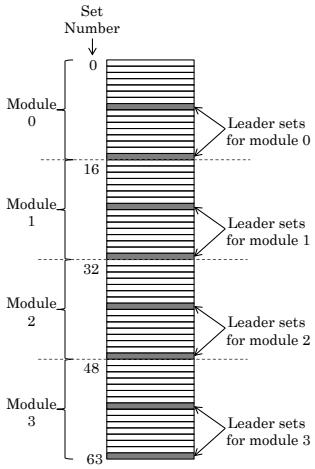

Fig. 3. Illustration of profiling information collection using ATD leader sets, assuming sampling ratio R_S = 8, Q = 4 modules and a total of 64 sets.

We now take a realistic example and assume that cache block size is 64B and associativity is 8 way. For a total size of 4MB, a cache will have 8192 sets and for Q = 16, each module will have 512 sets. For a sampling ratio of 64, the ATD will have 128 sets. Thus, 8 ATD sets correspond to each module. For meaningful sampling, we require that at least one ATD set should be there in each module, thus if S shows the number of L2 sets, then we require $\frac{S}{QR_S} > 1$. Hence, for Q = 16 and R_S = 64, we require $S > 1024$, which is true

for a LLC with cache size greater than 512KB. This condition easily holds for our experiments.

C. Taking Cache Reconfiguration Decision

We now show a simple analysis to derive the basis for turning-off a sub-way. Turning-off a sub-way for time length T saves leakage energy consumed in that sub-way, but it also increases the dynamic energy of cache and DRAM depending on the number of hits to that sub-way. Hence, a sub-way can be turned-off if the benefit from turning off is greater than the loss from turning-off. If H shows the number of hits to a sub-way, for turning-off a sub-way, we require[1]

$$\frac{P_{L2}^{Leak}T}{QW}(1-P_{off})(1+\Upsilon) > H(E_{L2}^{Dyn}+E_{DRAM}^{Dyn}) \quad (7)$$

Intuitively, the above equation shows that for small value of H, the dynamic energy (right hand side) saved by keeping a sub-way turned on will be smaller than the leakage energy (left hand side) saved by turning off the sub-way. Thus, the condition in Equation 7 is fulfilled when the number of hits to this sub-way is less than a threshold (α), given by

$$\alpha = \frac{P_{L2}^{Leak}(1-P_{off})(1+\Upsilon)T}{QW(E_{L2}^{Dyn}+E_{DRAM}^{Dyn})} \quad (8)$$

In this derivation, we have made some simplifying assumptions. We have neglected the effect of increase in execution time due to cache turnoff for sake of simplicity. Also, we assumed that a cache miss does not lead to a write-back. On taking these overheads into account, α will be further reduced since less number of misses can be tolerated. Since these effects are different for different workloads; instead of evaluating the overhead, we adopt a simpler approach: we simply scale the value of α by a constant factor $\lambda(\leq 1.0)$ to get the value of the threshold. Thus, α is given by

$$\alpha = \frac{P_{L2}^{Leak}(1-P_{off})(1+\Upsilon)T}{QW(E_{L2}^{Dyn}+E_{DRAM}^{Dyn})} \times \lambda \quad (9)$$

This approach makes the choice of M *application-independent* and hence, unlike the parameters used in few techniques (e.g. miss-bound in [5]), the need of offline profiling is avoided. The experimental results have shown that despite its simplicity, FlexiWay saves large amount of energy and outperforms conventional cache energy saving techniques. Also choice of λ also provides a knob for controlling aggressiveness of cache turn-off and thus offers flexibility to exercise trade-off between performance loss and energy savings.

V. Energy Saving Algorithm (ESA)

We now show the cache reconfiguration based energy saving algorithm of FlexiWay. The algorithm runs after a fixed interval size (e.g. 15M cycles). The algorithm collects counters to different recency positions in the LRU set-associative chain using ATD. In Algorithm 1, these are shown as L2Hit[0:$Q-1$][0:$W-1$].

[1]The energy-cost of block switching during turning off a sub-way is five orders of magnitude smaller than leakage energy saving, and hence, is ignored.

Algorithm 1: FlexiWay Energy Saving Algorithm

Input: L2Hit[0:$Q-1$][0:$W-1$]
Output: IsSubWayOn[0:$Q-1$][0:$W-1$] showing which sub-ways of different modules are turned-on.

```
Let IsSubWayOn[0:Q-1][0:W-1] denote the currently turned-on sub-ways of different modules
foreach module x = 0 to Q-1 do
    bool didChangeHappen = FALSE
    foreach sub-way v = W_min to W-1 do
        /* Look for turned-off sub-ways.                    */
        if IsSubWayOn[x][v] == FALSE then
            /* Sub-way v of module x is off. See if it can be turned-on.  */
            if L2Hit[x][v] > β then
                didChangeHappen = TRUE
                IsSubWayOn[x][v] = TRUE
                foreach sub-way e = 2 to v-1 do
                    if IsSubWayOn[x][v] == FALSE then
                        IsSubWayOn[x][v] = TRUE
                    end
                end
            end
        end
    end
    if didChangeHappen == FALSE then
        /* No sub-way could be turned on.                   */
        foreach sub-way v = W-1 to W_min do
            if IsSubWayOn[x][v] == TRUE then
                /* Sub-way v of module x is on. See if it can be turned off.  */
                if L2Hit[x][v] < α then
                    IsSubWayOn[x][v] = FALSE
                else
                    break
                end
            end
        end
    end
end
return IsSubWayOn[0:Q-1][0:W-1]
```

The algorithm works as follows. In each of the module, the algorithm first looks for turned-off sub-ways. If in the ATD, the number of hits seen in any sub-way is greater than β, then the sub-way can be turned on. To avoid (or reduce) oscillation, the threshold (β) for waking up a sub-way is set to be higher than the threshold for turning-off a sub-way. In our experiments, we set $\beta = \alpha + 50$. If a turned-off sub-way is turned-on, all sub-ways which are higher (nearer to MRU) in LRU chain are also turned-on. This is because we provision that all sub-ways, which are less recently used than a turned-off sub-way, must also be turned-off and vice-versa.

In a module, if no sub-way has been turned-on, then the algorithm looks for turning-off some sub-ways, using Eq. 9. However, if a sub-way is found which cannot be turned-on, the algorithm does not look further in that module for the same reason mentioned above. The algorithm executes only $\mathcal{O}(QW)$ steps and in each step performs a few comparisons and array look-ups, and thus its overhead is small.

VI. Implementation

Turning-off Cache Blocks: In this paper, we assume that cache blocks are turned-off using gated V_{dd} scheme [22]. This scheme inserts a "sleep" transistor between the ground (or supply) and the SRAM cells of the cache line. When this

transistor is off, the stacking effect of this transistor reduces the leakage current of SRAM cell to a near zero value [22]. We assume a specific implementation of gated V_{dd} (NMOS gated V_{dd}, dual Vt, wide, with charge pump) which reduces leakage energy by 97% and results in 5% area penalty and a negligible increase in cache access latency [22]. We have accounted for these overheads in our energy model.

Gated V_{dd} scheme has been used to achieve leakage control at both coarse-grain [5], [8], [10], [12] and fine-grain level [13], [21], [26]; and hence, it can be easily used for FlexiWay also. For each of the Q modules, we use $W - W_{min}$ control bits which control turning-off or turning-on of the sleep transistor of that sub-way. Note that the hardware functionality to turnoff cache blocks is already provided by state-of-the-art commercial chips [27], [28].

Counters: FlexiWay uses counters for recording the number of hits to W ways of all the Q modules. For example, for a typical 8-way cache with 16 modules and 64 bit counters, the total storage required for counters is only 1KB, which is very small. Since several processors already use counters for operating system [13], and their energy consumption is small compared to memory subsystem, we ignore the overhead of counters. Also note that FlexiWay does not require tables for offline profiling (unlike [6]) or per-block counters for recording application-ownership or access intensity (unlike [8], [13]). Also it does not require mapping table (unlike [12]) or changes to page table (unlike [29]).

Effect on Cache Access Time: FlexiWay provides fine granularity with caches of typical associativity and hence, does not require caches of large associativity which have higher access latency. Unlike for drowsy cache technique [18], with FlexiWay, reconfigurations happen only at the end of a large interval and hence, block switching does not lie on the critical access path. Also, unlike selective-sets or cache coloring (e.g. [11], [12], [30]), FlexiWay does not change the set-decoding and hence, it does not increase the cache access time.

Handling reconfigurations: When the number of ways is increased, the extra ways are simply turned-on. When the number of ways in reduced, the dirty blocks in ways to be turned off are written-back and valid blocks are discarded. Since reconfigurations happen infrequently, their overhead is easily amortized over the interval length.

VII. Experimental Methodology

A. Simulation Environment and Workload

We evaluate FlexiWay using Sniper, a state-of-the-art x86-64 multi-core simulator, which has been verified against real hardware [31]. A few parameters used in the experiments are shown in Table II. We use interval core model and each core has 128-entry ROB and a dispatch width of 4 micro-operations. The frequency is set to 2.2 GHz. The L2 cache is shared among the cores, while L1I and L1D caches are private to each core. All caches have a block size of 64B. Both L1D and L1I are 32KB, 4-way, LRU caches and have a latency of 2 cycles. The L2 cache is an 8-way, LRU cache with 12 cycle latency. The latency of main memory is 154 cycles and memory queue contention is also modeled. Interval length is 15M cycles, λ is taken as 0.75 and W_{min} is 2.

TABLE II
Some Parameters for Single, Two and Four-Core Systems

	N =1	N =2	N =4
Cache Size (MB)	2	4	8
E_{L2}^{Dyn} (nJ/access)	0.985	1.148	1.525
P_{L2}^{Leak} (Watt)	1.568	2.848	5.588
Number of Modules Q (for FlexiWay)	8	16	32
Memory Bandwidth (GB/s)	10	15	25

B. Evaluation Metrics

We show the results on percentage energy saved, weighted speedup [30], fair speedup [30] and cache ActiveRatio (defined as the average fraction of active cache blocks over entire execution [13]). Further, we show the results on absolute increase in L2 miss-per-kilo-instruction (MPKI). Through this, the increase in L2 misses resulting from cache turnoff and reconfigurations can be measured. We report *absolute* difference, instead of *relative* difference following previous works [11]. Across the workload, fair speedup and weighted speedup values are averaged using geometric mean of per-workload improvements and all other metrics reported in the paper are averaged using arithmetic mean.

C. Workloads

All benchmarks from 29 SPEC2006 suite with *ref* inputs are used as single-core workloads. Using these, 15 two-core and 15 four-core multiprogrammed workloads are created, such that except for completing the left-over group, each benchmark is used exactly once for two-core workloads and twice for four-core workloads. These workloads are shown in Table III.

TABLE III
Workloads Used in the Paper

Single-core workloads (SPEC2006 benchmarks) with their acronyms
As(astar), Bw(bwaves), Bz(bzip2), Cd(cactusADM), Ca(calculix)
Dl(dealII), Ga(gamess), Gc(gcc), Gm(gemsFDTD), Gk(gobmk)
Gr(gromacs), H2(h264ref), Hm(hmmer), Lb(lbm), Ls(leslie3d)
Lq(libquantum), Mc(mcf), Mi(milc), Nd(namd), Om(omnetpp)
Pe(perlbench), Po(povray), Sj(sjeng), So(soplex), Sp(Sphinx)
To(tonto), Wr(wrf), Xa(xalancbmk), Ze(zeusmp)
Two-core workloads: T1 to T15 (Using acronyms shown above)
T1(AsDl), T2(GcBw), T3(GmGr), T4(SoXa), T5(BzLq)
T6(OmLb), T7(NdCd), T8(CaTo), T9(SpMc), T10(LqMi)
T11(SjWr), T12(LsZe), T13(HmGa), T14(GkH2), T15(PePo)
Four-core workloads: F1 to F15 (Using acronyms shown above)
F1(HmSjPoH2), F2(SoGrZeGm), F3(OmSpLbGc), F4(BzGrLsMc)
F5(LsZeXaLq), F6(BwPoNdH2), F7(NdCaAsCd), F8(CaAsOmMc)
F9(BzDlGaGm), F10(SpGcLqHm), F11(XaLbMiGk), F12(SoNdMiBw)
F13(DlCdGkGa), F14(AsPeToWr), F15(SjToWrPe)

Each benchmark was fast-forwarded for 10B instructions and the workloads were simulated till each core in the workload completes at least 400M instructions. A core that

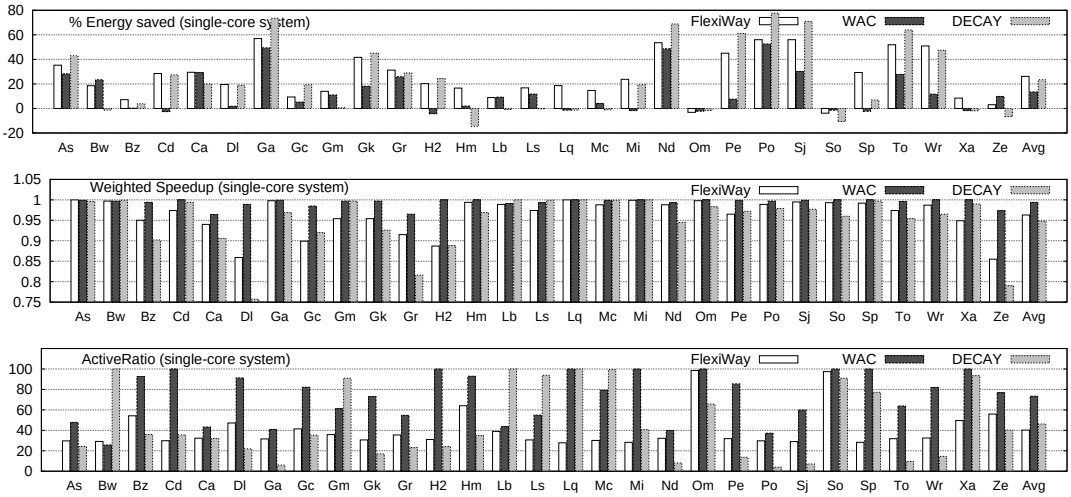

Fig. 4. Results for single-core system

has finished its 400M instructions continues to run, but for computation of weighted speedup and fair speedup, its IPC is recorded only for 400M instructions, following well-established simulation methodology [8], [32]. The value of energy is recorded for the entire execution, following [6], to comprehensively account for the effect of increased execution time on energy consumption and penalize a technique which harms performance or causes unfairness.

D. Comparison with Other Techniques

Way adaptable cache (WAC): WAC [10] uses selective-ways approach and keeps only few MRU (most recently used) ways of the cache active to save energy. WAC computes the ratio (Z) of hits to the least recently used *active* way and the MRU way. It also uses two threshold values T_1 and T_2. After every K cache hits, Z is computed. If $Z < T_1$, a single least-recently used cache way is turned-off. If $Z > T_2$, a single cache way is turned-on. Otherwise, no change is performed. Following Bardine et al. [10], we take the value of T_1 and T_2 as 0.005 and 0.02, respectively and K as 100,000.

Decay cache technique (DECAY): DECAY [13] turns off a cache line if it has not been accessed for the duration of 'decay interval' (DI). Following Kaxiras et al. [13], we use competitive algorithms theory to compute DI. From Section III and Table II, E_{DRAM}^{Dyn} = 70nJ and for single-core system with 2MB cache, P_{L2}^{Leak} = 1.568 Watt. Also, L2 has 32768 blocks and frequency is 2.2GHz. Thus, the ratio of DRAM access energy and L2 leakage energy per cycle per block is 70/(1.568/(2.2 $\times$ 32768)), which equals 3.2M cycles. This is taken as DI for single-core system. Similarly, DI for 2-core and 4-core systems are found, which are 3.5M and 3.6M cycles, respectively. Also, we implement DECAY using hierarchical counters, and decay both tag and data arrays [13].

We have chosen these techniques since they both use state-destroying leakage control like FlexiWay. Also, it helps us in evaluating how FlexiWay compares to both coarse-grain and fine-grain cache reconfiguration based techniques.

VIII. Results and Analysis

A. Main Results

Figure 4 and 5 show the results for single-core, 2-core and 4-core system configurations, respectively. For remaining quantities and results presented henceforth, we omit the per-workload figures due to space limitation and only state the average. The average value of increase in L2 MPKI with FlexiWay (WAC and DECAY) with single, dual and quad-core system are 0.71(0.10 and 1.01), 1.15(0.21 and 0.85), 1.40(0.14 and 1.07), respectively. The average value of fair speedup for dual and quad core system with FlexiWay (WAC and DECAY) are 0.96(0.99 and 0.96) and 0.95(0.99 and 0.96), respectively.

We now analyze the results. Firstly, FlexiWay outperforms both WAC and DECAY and provides nearly *double* the energy savings compared to WAC. To take decision about turning-off a way, WAC uses the *ratio* of number of hits to MRU and active LRU ways. Thus, even if the actual number of hits are small, WAC keeps a way active only based on the ratio. In contrast, FlexiWay uses the *absolute* value of hits to a sub-way and hence, it can more effectively turn-off the sub-ways with small number of cache hits. In any interval, WAC turns-off or turns-on only one cache way, while FlexiWay may turn-off or turn-on up to $W - W_{min}$ sub-ways of a single module, for all the modules. Thus, FlexiWay adapts to the changing working set of the program much more quickly and thus, can save larger amount of energy.

While DECAY is effective for L1 cache, its effectiveness reduces greatly when applied to L2 cache [26], since, actually, the L2 observes the behavior of the L1 *misses* and hence, generational behavior of cache lines is less apparent in L2. In contrast, FlexiWay works well for L2 which spend large fraction of energy in the form of leakage energy. Also, it has been shown that the optimal value of decay interval (DI) varies widely with different workloads [33]. Since multicore systems may run arbitrary combinations of benchmarks, finding optimal value of DI becomes more difficult in multicore systems.

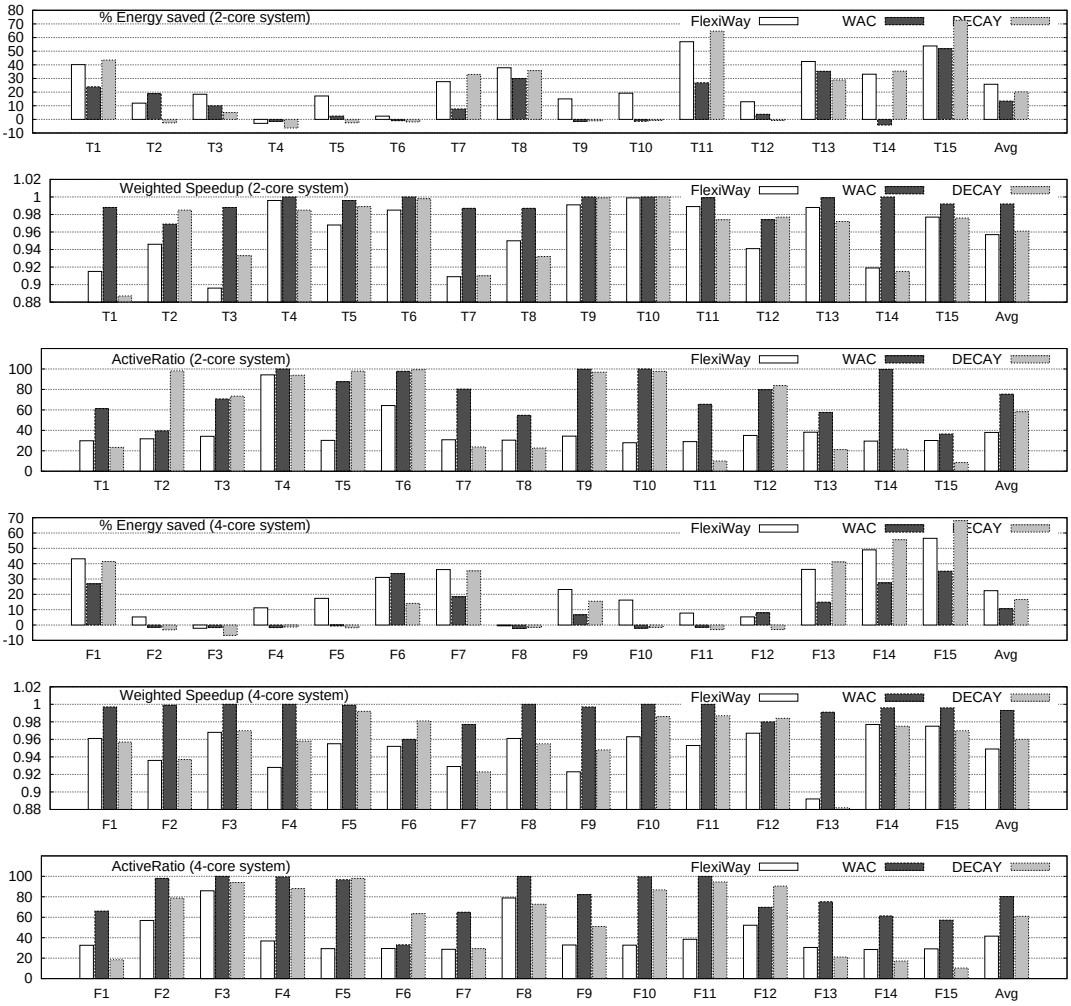

Fig. 5. Results for dual-core and four-core system

Both FlexiWay and WAC guarantee a minimum associativity to the programs and keep those MRU (and near MRU, depending on W_{min}) blocks always ON which are highly likely to be accessed again. This is especially important in LLC, since off-chip accesses are very costly. In contrast, DECAY attempts to turn-off all the blocks and does not directly take advantage of the set-associative structure of cache.

From the ActiveRatio values, we conclude that FlexiWay turns off larger fraction of cache than other techniques. For some workloads, e.g., Lq(libquantum) and Mi(milc), WAC fails to perform any reconfiguration, since they have near zero hit rates. For workloads which use L2 intensely, hardly few cache blocks stay idle for the duration of DI and hence, for such workloads, DECAY does not effectively reconfigure the cache, e.g. Lq, T2(gcc,bwaves), T6(omnetpp,lbm) etc. Per-workload adaptation of DI would be required for improving energy savings of DECAY in such workloads.

For several workloads, the hits in cache are uniformly distributed to different ways and hence, WAC does not perform any cache reconfiguration, for example, Cd (cactusADM), T4(soplex,xalan), F3(omnetpp,sphinx,gemsFDTD,gcc) etc. Thus, FlexiWay leverages the opportunity of fine-grain cache reconfiguration better than WAC. DECAY uses much finer reconfiguration and hence saves larger energy in some workloads than FlexiWay; however, it also leads to very aggressive cache reconfiguration which results in poor worst-case performance and energy saving in some workloads, such as So(soplex) and T4 etc.

In terms of weighted and fair speedup, FlexiWay and DECAY are close and WAC provides better results. Also, L2 MPKI increase with WAC are less than that using other techniques. However, this is partially due to the fact that WAC turns-off very small fraction of cache. Also, the increase in DRAM energy caused by FlexiWay is more than compensated by the energy saving achieved in the cache. Further, as we show in Section VIII-B, by controlling the parameters such as λ and W_{min}, the aggressiveness of cache reconfiguration of FlexiWay can be controlled which reduces the performance loss at the cost of small reduction in energy saving.

With FlexiWay, the average value of fair speedup is very

close to that of weighted speedup and hence, FlexiWay does not cause unfair slow-down of any application.

B. Parameter Sensitivity Study

We hereafter focus only on FlexiWay and study its sensitivity to different parameters. Each time we only change one parameter from default values and summarize the results in Table IV. For brevity, we only show values of energy saving and weighted speedup.

TABLE IV
RESULTS FOR DIFFERENT PARAMETERS. DEFAULT VALUES: EMBEDDED ATD, R_S = 64, INTERVAL = 15M CYCLES, λ = 0.75 AND W$_{min}$=2

	% Energy Saved			Weighted Speedup		
	N=1	N=2	N=4	N=1	N=2	N=4
Default	26.26	25.74	22.39	0.96	0.96	0.95
Separate ATD	26.38	26.20	22.80	0.96	0.96	0.95
$R_S = 128$	26.01	25.81	22.48	0.96	0.96	0.95
Interval=10M	26.35	25.80	22.24	0.96	0.96	0.95
Interval=20M	25.79	25.40	22.39	0.96	0.96	0.95
$\lambda = 0.5$	25.97	24.76	21.76	0.97	0.96	0.96
$\lambda = 1.0$	25.95	25.61	22.19	0.96	0.95	0.94
W_{min}=3	23.00	23.13	20.24	0.97	0.97	0.97

Using separate ATD: We compute the energy values of separate ATD using CACTI and include the energy consumption of ATD in E_{Algo}. For sake of brevity, we omit these energy values. Clearly, use of separate ATD provides energy savings comparable to that with embedded ATD. Thus, the separate ATD can be used as an alternative to embedded ATD.

Change in Sampling Ratio: On changing R_S to 128, we still achieve large energy savings. Thus, R_S can be increased to reduce the overhead of FlexiWay without reducing energy savings or harming performance.

Change in Interval Size: Smaller interval size allows aggressive cache reconfiguration, which enables saving larger energy in some workloads (e.g. namd) while reducing the energy saving in other workloads (e.g. soplex). Similar trends are also observed for the larger interval size. However, on average, the energy savings and performance are only slightly affected. This shows the FlexiWay works well for an interval length in the range of a few million cycles.

Change in λ: On changing λ to 0.5, energy savings are reduced, but performance is improved. On changing λ to 1.0, performance is slightly reduced, which also leads to reduction in energy savings. Thus, value of λ near 0.75 show a conservative value and it can be further reduced to improve performance at the cost of reduction in energy saving.

Change in W_{min}: On increasing W_{min} to 3, the performance is improved at the cost of energy savings, although the energy savings are still large. Thus, by changing W_{min}, a balance between energy saving and performance loss can be achieved.

IX. CONCLUSION

In this paper, we presented FlexiWay, a technique which uses fine-grain cache way-based turnoff for saving cache leakage energy. FlexiWay logically divides the cache into several modules and turns-off cache at the granularity of a single way of a module. The experimental results performed using single, dual and quad-core systems have shown that FlexiWay is effective in saving energy and incurs only small loss of performance.

REFERENCES

[1] S. Borkar and A. Chien, "The future of microprocessors," *Communications of the ACM*, vol. 54, no. 5, pp. 67–77, 2011.
[2] http://www.anandtech.com/show/5091/intel-core-i7-3960x-sandy-bridge-e-review-keeping-the-high-end-alive.
[3] S. Rodriguez *et al.*, "Energy/power breakdown of pipelined nanometer caches (90nm/65nm/45nm/32nm)," in *ISLPED*, 2006.
[4] C. Zhang *et al.*, "A highly configurable cache architecture for embedded systems," in *ISCA*, 2003, pp. 136–146.
[5] S. Yang *et al.*, "An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-caches," in *HPCA*, 2001.
[6] W. Wang *et al.*, "Dynamic cache reconfiguration and partitioning for energy optimization in real-time multicore systems," in *DAC*, 2011.
[7] D. H. Albonesi, "Selective cache ways: on-demand cache resource allocation," in *MICRO*, 1999, pp. 248–259.
[8] K. Sundararajan *et al.*, "Cooperative partitioning: Energy-efficient cache partitioning for high-performance CMPs," *HPCA*, 2012.
[9] I. Kotera *et al.*, "Power-aware dynamic cache partitioning for CMPs," *Transactions on HiPEAC*, pp. 135–153, 2011.
[10] A. Bardine *et al.*, "Way adaptable D-NUCA caches," *IJHPSA*, vol. 2, no. 3, pp. 215–228, 2010.
[11] S. Mittal *et al.*, "EnCache: Improving cache energy efficiency using a software-controlled profiling cache," in *IEEE EIT*, May 2012.
[12] ——, "CASHIER: A Cache Energy Saving Technique for QoS Systems," *26th IEEE VLSID*, 2013.
[13] S. Kaxiras *et al.*, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *ISCA*, 2001.
[14] D. Rolán *et al.*, "Adaptive line placement with the set balancing cache," in *MICRO*, 2009, pp. 529–540.
[15] M. K. Qureshi *et al.*, "The V-Way cache: demand-based associativity via global replacement," in *ISCA*, 2005, pp. 544–555.
[16] J. Abella and A. González, "Heterogeneous way-size cache," in *ICS*, 2006, pp. 239–248.
[17] D. Benitez *et al.*, "A reconfigurable cache memory with heterogeneous banks," in *DATE*, 2010, pp. 825–830.
[18] K. Flautner *et al.*, "Drowsy caches: simple techniques for reducing leakage power," in *ISCA*, 2002.
[19] L. Li *et al.*, "Soft error and energy consumption interactions: a data cache perspective," in *ISLPED*. IEEE, 2004, pp. 132–137.
[20] ——, "Leakage energy management in cache hierarchies," *PACT*, pp. 131 – 140, 2002.
[21] H. Hanson *et al.*, "Static energy reduction techniques for microprocessor caches," *IEEE TVLSI*, 2003.
[22] M. Powell *et al.*, "Gated-Vdd: a circuit technique to reduce leakage in deep-submicron cache memories," in *ISLPED*, 2000, pp. 90 – 95.
[23] CACTI 5.3, http://quid.hpl.hp.com:9081/cacti/.
[24] H. Zheng *et al.*, "Decoupled DIMM: building high-bandwidth memory system using low-speed DRAM devices," in *ISCA*, 2009.
[25] M. K. Qureshi *et al.*, "A case for MLP-aware cache replacement," in *ISCA*, 2006, pp. 167–178.
[26] J. Abella *et al.*, "IATAC: a smart predictor to turn-off L2 cache lines," *ACM TACO*, vol. 2, no. 1, pp. 55–77, 2005.
[27] N. Kurd *et al.*, "Westmere: A family of 32nm IA processors," in *ISSCC*, 2010, pp. 96–97.
[28] A. Naveh *et al.*, "Power and thermal management in the Intel Core Duo processor," *Intel Technology Journal*, 2006.
[29] T. Sherwood, B. Calder, and J. Emer, "Reducing cache misses using hardware and software page placement," *Proceedings of the 13th international conference on Supercomputing*, pp. 155–164, 1999.
[30] J. Lin *et al.*, "Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems," *HPCA*, 2008.
[31] T. E. Carlson *et al.*, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations," in *SC*, 2011.
[32] M. Qureshi *et al.*, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," *MICRO*, 2006.

[33] H. Zhou *et al.*, "Adaptive mode control: A static-power-efficient cache design," *ACM TECS*, 2003.