Dynamic Associative Caches: Reducing Dynamic Energy of First Level Caches

Karthikeyan Dayalan, Meltem Ozsoy, Dmitry Ponomarev
State University of New York at Binghamton
Email: kdayala1,mozsoy,dima@cs.binghamton.edu

***Abstract*—We propose Dynamic Associative Cache (DAC) - a low complexity design to improve the energy-efficiency of the data caches with negligible performance overhead. The key idea of DAC is to perform dynamic adaptation of cache associativity - switching the cache operation between direct-mapped and set-associative regimes - during the program execution. To monitor the program needs in terms of cache associativity, the DAC design employs a subset of shadow tags: when the main cache operates in the set-associative mode, the shadow tags operate in the direct-mapped mode and vice versa. The difference in the hit rates between the main tags and the shadow tags is used as an indicator for the cache mode switching. We show that DAC performs most of its accesses in the direct-mapped mode resulting in significant energy savings, at the same time maintaining performance close to that of set-associative L1 D-cache.**

I. Introduction

First-level caches consume a significant amount of energy in modern processors. According to recent reports [13], [11], between 12% and 45% of the processor core energy is attributed to first level caches. Most processors today use set-associative first-level caches to increase the cache hit rates. In an N-way cache, all N tag and data arrays are accessed in parallel, but only one of the ways return the requested data on a cache hit - the rest of the ways are accessed in vain resulting in energy wastage. Serializing tag and data access and only reading the data from a matching way is impractical for first-level caches, as it increases the critical timing path or requires an additional pipeline stage for the cache access, thus affecting performance.

In this paper, we propose Dynamic Associative Cache (DAC) - a technique that dynamically adjusts the cache associativity (between direct-mapped and set-associative mode) in response to program demands. DAC is based on the key observation that only a relatively small number of programs substantially benefit from a higher cache associativity, but the majority of programs perform well even with direct-mapped caches.

Without compromising the cache size, the DAC design implements direct-mapped style access to a set-associative cache by using a few least significant bits of the tag to explicitly select a cache way to be accessed - the access to all other ways is disabled, thus saving energy. In a sense, the combination of the index bits and least significant tag bits serve as an index to the direct-mapped cache. When DAC operates in a set-associative mode, the regular access to all cache ways is performed.

The goal of DAC is to minimize the number of accesses that are performed in the set-associative mode (thus saving energy) without sacrificing performance. This is achieved by carefully controlling the mode transitions and performing these transitions based on the triggers generated by the cache performance monitoring unit (CPMU). CPMU is a simple logic that employs additional *shadow tags* [16] to keep track of how the cache would have performed if it was operating in the other mode. For example, during the time when the cache is operating in the set-associative mode, the shadow tags track its hypothetical performance in the direct-mapped mode. Conversely, if the cache operates in the direct-mapped mode, the shadow tags track the performance of a hypothetical set-associative cache. The CPMU logic measures the difference in the hit rate between the actual and the hypothetical modes and performs mode transitions as necessary, when certain thresholds are crossed. Compared to a simple use of cache misses as the mode switch trigger, such shadow tag monitoring allows to distinguish conflict misses from the capacity misses. Periodically, all monitoring counters are reset. In order to reduce the design complexity due to the shadow tags, we utilize the idea of *set sampling* [8], where only a small subset of the cache sets is monitored through the shadow tags. Access to all other sets, not covered by the selected shadow sets are not accounted for making transition decisions.

A transition from direct-mapped to set-associative access mode in DAC is simple and does not require any additional actions. This is because all data brought into the cache while it was operating in the direct-mapped mode will be found during the set-associative search of the tags. However, the reverse transition from set-associative to direct-mapped mode is more challenging. Indeed, the direct-mapped search would be limited to only one way, and therefore some cache hits can be erroneously identified as misses. Furthermore, it can open a possibility of having duplicate data in the cache, once the erroneous miss is serviced. To avoid these problems, several solutions are possible. For a write-through cache, the entire cache can be simply invalidated when a transition from set-associative to direct-mapped mode is made. For a write-back cache, the mode transition would require the write-back of all dirty lines to the L2 cache.

Since the transition from set-associative to direct-mapped mode is more complex, we propose two variants of DAC: *DAC-Budget*, and *DAC-Deluxe*. *DAC-Budget* implementation only supports the transition from direct-mapped to set-associative mode and operates in the set-associative mode

until the process is context switched out by the operating system following a timing interrupt. After getting rescheduled on the CPU following a context switch, the process will again begin operating with the L1 caches accessed in direct-mapped style. Of course, in this case the invalidation or the write-back of all dirty lines in the cache would still need to be performed. but continuous switching between the modes during the single time quantum is not supported. Since the L1 caches typically do not survive context switches anyway, cache invalidations would not have an impact on performance. Even if some cache state survives a normal context switch, this state would be quickly re-fetched from the L2 cache in DAC. *DAC-Deluxe* implementation, in contrast, allows bidirectional mode transition even during a single time quantum, thus the CPMU is modified to support it.

The main contributions and the key results of this paper are as follows:

- We propose DAC - a cache design that is capable to dynamically change its associativity during the execution without turning off any cache lines/ways and effectively utilizing the entire available cache space for storing application data.
- We evaluate DAC using cycle-accurate processor simulator [4] and CACTI tool [5] for measuring energy reduction and estimating the impact on the cache access time.
- We show that DAC saves around half of the dynamic energy in the L1 data cache, maintains MPKI close to that of the set-associative cache, and increases the cache access time by less than 2%.

II. Motivation for DAC

The DAC design was motivated by the observation that a significant number of benchmarks in the SPEC 2006 suite [14] didn't benefit from the increased first level data cache associativity and achieve similar levels of performance even with direct-mapped caches. Figure 1 depicts this trend for D1-cache. The Y axes shows the number of Misses per Kilo Instructions (MPKI) for two cases: direct-mapped cache and 4-way set-associative cache of the same size. On the average, the MPKI is reduced by 30% when set-associative caches are used instead of direct-mapped caches. At the same time, the majority of benchmarks show very little difference, and the average impact is mostly due to a few selected benchmartks (*dealII*, *gamess*, *hmmer*, *namd*, *povray* and *sphinx3*).

The goal of the DAC design is to achieve higher energy-efficiency of caches by allowing the caches to operate mostly in the direct-mapped mode, and only switch to the set-associative mode when it is determined that higher associativity would significantly improve cache performance.

III. DAC Design Details

The DAC logic is implemented as a series of simple additions to the traditional virtually-indexed physically-tagged (VIPT) set-associative cache. In the set-associative mode, accesses to DAC are performed in exactly the same manner as in

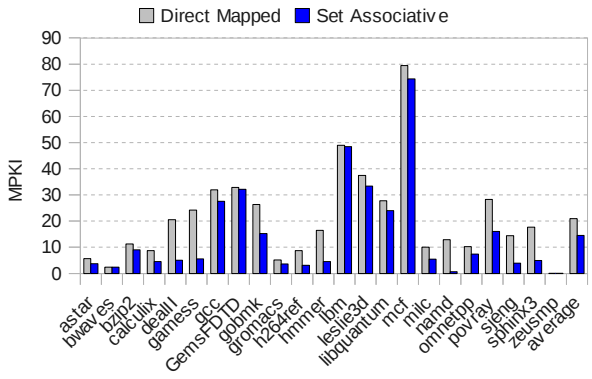

Fig. 1. Misses Per Kilo Instructions for Set associative(32KB) and Direct Mapped Data caches

traditional set-associative caches, with no changes. However, to implement a direct-mapped style access in DAC, a new indexing scheme is needed. Specifically, we propose to use few least significant bits of the tag (2 bits for a 4-way set associative cache) to explicitly select a specific way in the selected set that should be accessed. All other ways are not energized in this cycle, thus saving the dynamic energy incurred during the read out of the tag and data arrays, as well as the energy of tag comparators and sense amplifiers associated with the non-selected ways. Figure 2 shows the schematic of DAC and also demonstrates how the disabling of some cache ways can be implemented.

A. Hardware Modifications for DAC

To implement DAC, we need a mechanism to determine when an application would benefit from higher associativity and when direct-mapped cache configuration is sufficient for maintaining good performance. In order to accomplish that, we propose to dynamically measure the cache performance in both modes of operation by employing the idea of shadow tags [16]. Specifically, when the DA cache operates in a particular mode, the shadow tags track the hypothetical cache hit rates in the other operation mode, using the replacement policy of that mode. To reduce the hardware complexity and the area overhead of maintaining shadow tags, we rely on set sampling mechanism [8], where we only track a few selected sets that act as representatives of the entire cache. Previous work showed that the set sampling idea works well in various designs that require dynamic monitoring of cache behavior [8]. Our results also demonstrate that the lack of precision due to set sampling leads to minimal loss in performance compared to the design with full set of shadow tags, because the set usage across the L1 cache is relatively uniform. The caches that we study in this paper have 128 sets, and we sample every 16th set resulting in 8 sets of shadow tags.

In addition, the DAC design uses a counter that tracks the differences between the cache hit rate in the two modes. If the value of the counter crosses a predefined threshold, then the cache mode transition is triggered - the specific algorithm is described in the next subsection. Finally, DAC needs a logic

to disable the access to all but one cache ways if the access is performed in the direct-mapped mode. This logic is illustrated in Figure 2.

B. The DAC Mode Transition Algorithm

As described in Section 1, DAC can be implemented either in a simpler way (where only the transition from direct-mapped to set-associative mode is controlled, and at the granularity of context switches the cache is reset back to the direct-mapped mode), or in a more complex way, where the transitions between both modes are supported even between consecutive context switches.

First, we describe the simple algorithm (which we call *DAC-Budget*). In the DAC-Budget design, the cache begins execution in the direct-mapped mode. At the same time, the replacement logic in the shadow tags is setup so that the shadow tags perform in the set-associative regime. We also keep a counter that we call *SWITCH* that is initialized to zero. Every time that an access is performed to one of the sets that is tracked by the shadow tags (all other set accesses are ignored for the purposes of mode switching algorithm), the cache hit and miss outcomes in the main and the shadow tags are used to update the *SWITCH* counter value as follows. If both main tags and shadow tags indicate a hit or a miss, then the value of the counter stays the same. If the main tags indicate a miss, but the shadow tags indicate a hit, the counter value is incremented. If the main tags indicate a hit, but the shadow tags indicate a miss, then the value of the counter is decremented. Essentially, the value of *SWITCH* indicates how many extra misses that the DA cache encountered in the direct-mapped mode would have been hits if the cache was operating in the set-associative mode.

If the *SWITCH* counter value crosses a predetermined threshold (called *GO_SA*), then the transition to the set-associative mode is performed. This particular mode transition does not require any special action from the cache, all tag and data arrays are now read and a more flexible cache placement is enabled. In addition, the counter value is periodically reset to zero if the threshold value has not been crossed during this period. This captures the situations where the benchmark enters a long phase with no difference in the cache hit rate between direct-mapped and set-associative caches. After such a phase ends, resetting the counter guarantees that the counting to the threshold will begin from zero.

Once the decision to switch to the set-associative mode is made, the cache stays in that mode until the expiration of the time slice allocated by the OS to this process. Upon the context switch, the cache is returned to the direct-mapped mode, all cache contents are invalidated and the dirty lines are written back (assuming the write-back cache; simple invalidations are sufficient for the write-through caches). Since the contents of the L1 caches do not typically survive context switches anyway, the additional performance impact is minimal. Most of the data will be quickly brought back from the L2/L3 caches upon the rescheduling of the process. Figure 3-a summarizes this algorithm.

A more complex, but also a more dynamic scheme is to support transitions between the modes in both directions. In this scheme, which we refer to as *DAC-Deluxe*, after a transition to the set-associative mode is performed, it is also possible to revert the cache operation back to the direct-mapped mode before the end of the process time slice. To achieve that, once the main cache is switched to the set-associative mode, the shadow tags start operating in the direct-mapped mode. During the switch to the direct-mapped mode, the consistency of the shadow tags will be impacted (as is the case for the main cache), but this has no implications on correctness because the shadow tags are only used for guiding the mode switching. Even if the hits/misses determined by the shadow tags are slightly inaccurate, it will only impact the switching time, and thus the extent of energy saving. For this reason, we do not invalidate the shadow tags when they switch to the direct-mapped access mode.

After the main cache switches to the set-associative mode and the shadow tags switch to the direct-mapped mode, the *SWITCH* counter is reset to zero. On every access to a set that is sampled by the shadow tags, the counter is incremented if the main set associative cache indicates a hit and the shadow direct-mapped tags indicate a miss. In addition, the counter is decremented if the shadow tags indicate a hit, but the cache tags indicate a miss. The second threshold, called *TO_DM*, is

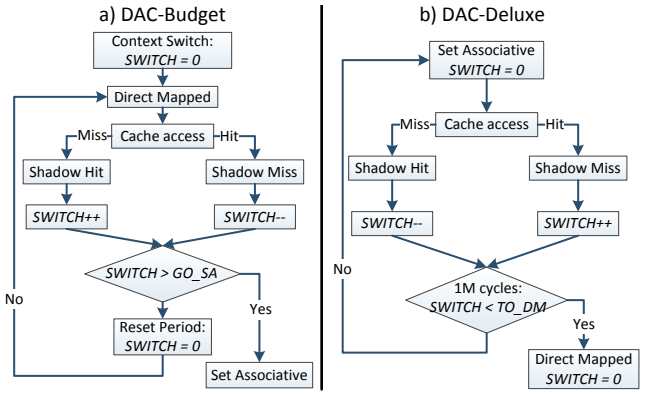

Fig. 3. DAC Mode Transition, DAC-Budget and DAC-Deluxe

used in *DAC-Deluxe* scheme to control the switching back to the direct-mapped mode. However, the logic in this case is different from the case of switching to set-associative mode. Here, the idea is to switch back to the direct-mapped mode if the counter value is relatively small, indicating little difference between the hit rates of the two cache access modes.

We implement this transition as follows. Periodically (every 1 Million cycles in our experiments) we check if the value of the *SWITCH* counter has crossed the *TO_DM* threshold. If it did (indicating significant advantage of set-associative cache over the direct-mapped cache in this interval), no further action takes place other than resetting the counter value to zero for the next interval. Otherwise, if the threshold has not been crossed, the transition back to the direct-mapped mode is triggered. This situation indicates that the difference between the hit rates

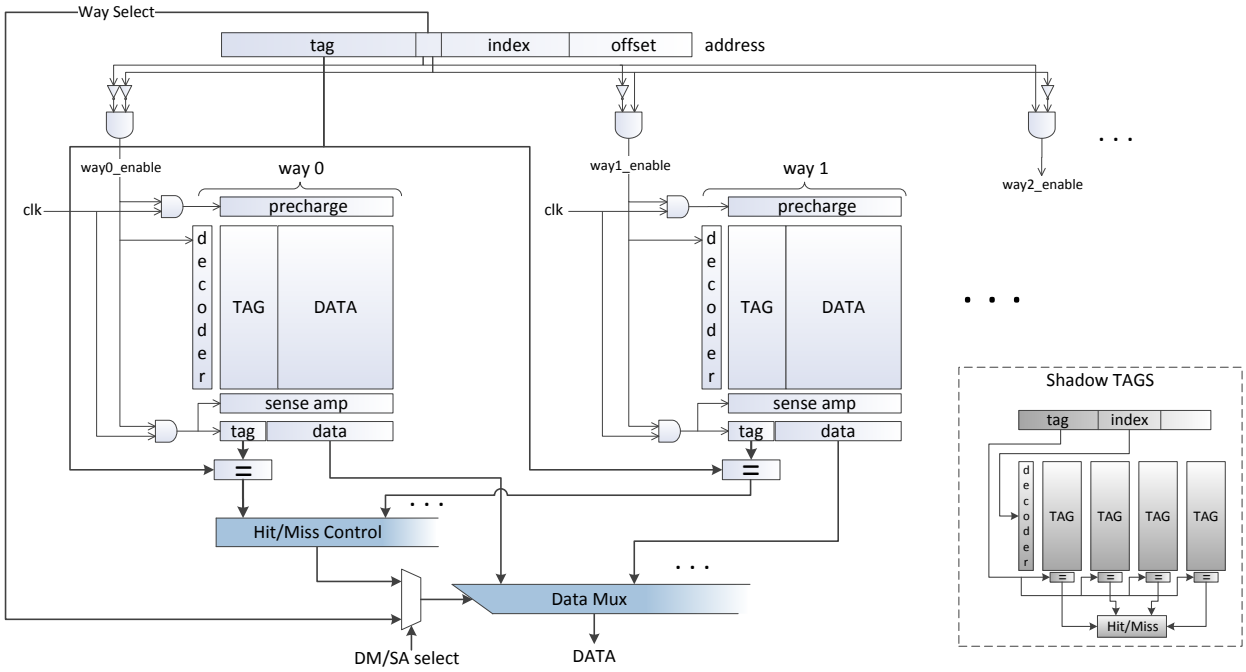

Fig. 2. DAC design for a 4-way set-associative cache (only ways 0 and 1 are shown)

in the two cache access modes was insufficient to warrant operation in the set-associative mode.

Of course, the performance/energy trade-offs of this algorithm depend on the accurate choice of the above mentioned thresholds. If we select the thresholds very small, then most of the benchmarks will shift back to set-associative and will remain over there or keep on shifting to both modes repeatedly. In the other case, if we select the threshold to be too high, then the benchmarks won't shift to set-associative and remains in direct-mapped mode for entire simulation. Eventhough, we get more power savings, this will affect the performance of the benchmarks. so, if power is the main concern for the application with some performance loss, then we can select high threshold value or if performance is the main concern, then we can select the low threshold value. The best threshold value will give optimal power savings with less performance loss. The values mentioned in this section were determined to perform the best in our simulations, therefore we use them in the result section.

C. Handling Synonyms

The baseline cache that we assume for this study is virtually-indexed and physically-tagged. Therefore, the synonyms (or aliases — multiple virtual addresses corresponding to the same physical address) do not present a problem to the baseline design, as the addressing is basically performed based on physical addresses. However, in the DAC design, the least significant tag bits (the ones used for the way selection in the direct-mapped mode) can generally change from virtual to physical addresses, thus resulting in synonym problem. This is similar to any virtually-addressed cache, but the difference is that in the DAC design only a few least significant tag bits (e.g. 2 bits for a 4-way cache) can cause the problem.

There are two solutions to this aliasing problem. First, if the operating system ensures that the bits used to select the cache way do not change in the course of address translation between virtual and physical address, then the problem does not exist. Modern operating systems routinely support such page mapping restrictions (not changing a few least significant bits of the page number during translation) in order to support larger caches in a system that features parallel access to the cache and the TLB. If such support is in place, it can be utilized by the DAC design directly.

Alternatively, we can exploit the observation that even if the synonyms exist in DAC, they will only be localized within the same set of the cache if the cache is viewed as a set-associative cache (all aliases will have the same virtual and physical index bits). Therefore, to detect synonyms we need to check all of the tags of the selected set. A small downside here is that we will only save the energy of the data arrays, and will need to energize all tag arrays. Since the cache energy is dominated by the data arrays, this is a relatively small sacrifice. In summary, the aliasing problem in DAC can be handled either for free (if the OS supports it), or with modest energy cost by accessing all cache tags in the selected set on every access.

IV. Evaluation Methodology

In order to determine the performance impact of the cache architecture described in this paper, we used M-Sim 3.0 [4] - a significantly modified version of the Simplescalar 3.0d simulator. The simulated processor configuration is depicted in Table I. For our studies we used 23 SPEC CPU2006 [14]

TABLE I
CONFIGURATION OF THE SIMULATED PROCESSOR

Parameter	Configuration
Machine Width	8-wide fetch, issue and commit
Window Size	128-entry ROB, 48-entry LSQ and Issue Queue
Physical Registers	128 Integer + 128 FP Physical Registers
L1 I-Cache	32 KB, 4-way set-associative, 64 byte line, 1 cycle hit time
L1 D-Cache	32 KB, 4-way set-associative, 64 byte line, 1 cycle hit time
L2 Unified Cache	512 KB, 8-way set-associative, 128 byte line, 10 cycle hit time
Memory latency	300 cycles

benchmarks. The benchmarks were compiled on a native Alpha AXP machine running tru64 Unix operating system. For each benchmark, we simulated 500 million instructions after skipping the initial 2 Billion instructions to avoid simulating the initialization stages. For measuring cache energy reduction and circuit delay, we used CACTI 6.5 [5]. Using CACTI, we created a direct-mapped, a 4-way set-associative, and DAC caches with total cache size of 32 KB in each case and the block size of 64 bytes. We used 4-way banked caches and 45 nm technology. The number of optimal sub-arrays determined by CACTI with the maximum frequency and minimum power design objective is shown in Table II. The data arrays of the direct mapped cache and DAC are divided into 8 sub-arrays with 4 segments for wordlines(Ndwl) and 2 segments for bitlines(Ndbl). The data array of the set-associative cache is divided into 16 sub-arrays. We summarized the energy usage per access and leakage power per bank for all cache designs in Table II. These energy values are used to calculate the efficiency of the cache schemes in Section V. Note that DAC saves energy in direct mapped mode, otherwise it consumes same amount of energy as a set-associative cache and shadow tags (only a small portion of tag array).

TABLE II
CACTI CONFIGURATION FOR DIRECT-MAPPED AND SET-ASSOCIATIVE CACHES

	Direct Mapped (32KB)	4-way Set Associative	DAC (in DM mode)
Energy/access	0.16618(nJ)	0.39985(nJ)	0.15457(nJ)
Leakage/bank	28.4582(mW)	64.5997(mW)	19.7505(mW)
Ndwl/Ntwl	4/2	8/2	4/2
Ndbl/Ntbl	2/2	2/2	2/2

We also evaluated the access time of the new cache design using CACTI. In order to calculate access time, we modified CACTI source code to integrate the Direct-Mapped/Set-Associative selection multiplexer and the way select decoder into the design.

V. RESULTS AND DISCUSSIONS

In this section, we present the results of performance, Misses Per Kilo Instructions and energy evaluation of the DAC design. For the results presented in this section we assumed 8 sets of shadow tags (that is, every sixteenth set is sampled), the value of *GO_SA* is set to 1024 for the *DAC-Budget* and *DAC-Deluxe* scheme and the value of *TO_DM* is set to 1024 for the *DAC-Deluxe* scheme. The counter was reset to zero for every 1 million cycle.

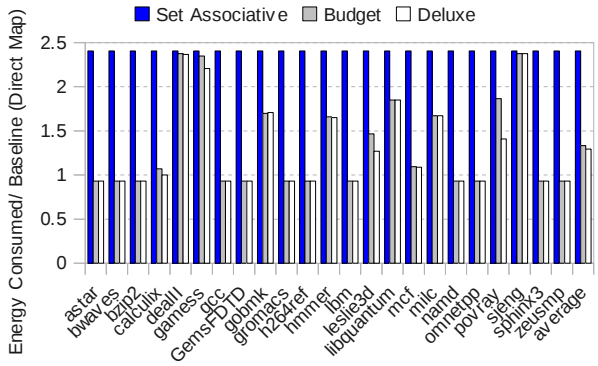

Fig. 4. Energy of DAC and Set-Associative Data Cache normalized to Direct-Mapped Cache Energy

Figure 4 shows the dynamic energy of set-associative data cache and the two DAC schemes. All values are normalized to the energy of the baseline direct-mapped data cache. The set-associative cache expends more than twice the dynamic energy of the direct-mapped cache of the same size. The DAC-Budget design consumes about 33% more energy than the direct-mapped cache and 45% less energy than set-associative, while the DAC-Deluxe scheme consumes 30% more energy than the direct-mapped cache and 47% less energy than set-associative cache. Using any DAC mode(*Budget* or *Deluxe*), one can save at least 80% of the energy that would be wasted with a set-associative cache. The energy consumption between the *DAC-Budget* and *DAC-Deluxe* almost remains same, except for the benchmarks which keeps on shifting back and forth between direct-mapped and Set-associative mode. In this graph, the benchmarks such as *calculix*, *gamess*, *leslie3d*, *povray* shifts from direct-mapped mode to set-associative mode in both *DAC-Budget* and *DAC-Deluxe* scheme. But in *DAC-Deluxe* scheme, these benchmarks shifts back to direct-mapped mode from set-associative mode. Hence these benchmarks spend less time in set-associative mode in *DAC-Deluxe* scheme compared to *DAC-Budget* scheme. So, the energy saved by *DAC-Deluxe* is more than the energy saved by the *DAC-Budget* scheme.

Figure 5 shows the percentage of access for DAC performed in the direct-mapped mode. The *DAC-Deluxe* scheme stays in the direct-mapped mode almost all of the time for most of the benchmarks. Only *dealII*, *gamess* and *sjeng* operate mostly in set-associative mode. This is in line with the results presented in the motivation section, as the performance of these benchmarks truly benefit from higher associativity. For the *DAC-Budget* scheme, most of the benchmarks switch at some point to the set-associative mode, and they stay in it for remaining simulation time. Whereas in *DAC-Deluxe* scheme,

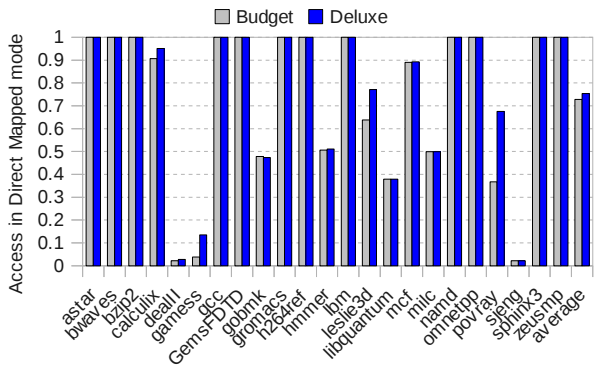

Fig. 5. Percentage of D-Cache Accesses Performed in Direct-Mapped Mode

the benchmarks such as *gamess*, *leslie3d* and *povray* moves back to direct-mapped mode from set-associative mode when there is no benefit in set-associative mode. So, the *DAC-Deluxe* scheme spends more time in direct-mapped mode compared to *DAC-Budget*.

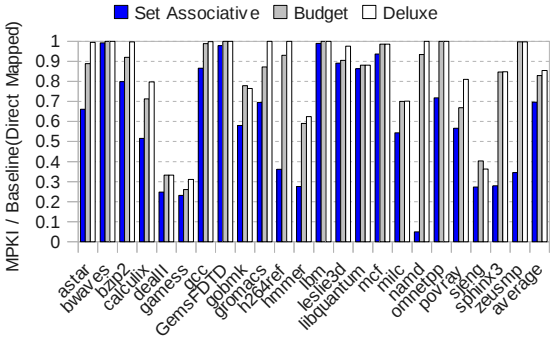

Fig. 6. MPKI for DAC and Set-Associative Cache normalized to MPKI of Direct-Mapped Cache

The impact of the DAC schemes on the cache misses is depicted in Figure 6. The results are normalized to the MPKI of the baseline direct-mapped cache. While some benchmarks such as *bwaves*, *GemsFDTD*, *lbm* experience the same amount of misses with DAC as with direct-mapped cache, other benchmarks (*dealll*, *gamess*, *milc*, *sjeng*) have significantly less misses with DAC compared to direct-mapped cache and approach the number of misses experienced by set-associative cache. On average, the DAC designs cover about half of the MPKI gap between the direct-mapped and set-associative caches.

Finally, Figure 7 shows the performance impact of both the *DAC-Budget* and the *DAC-Deluxe* scheme. All IPC values are normalized to the baseline IPC of the set-associative cache (highest performing design). We observe that *bwaves*, *zeusmp* benchmarks match the set-associative performance even with direct-mapped cache. However, other benchmarks, such as *gamess*, *povray*, *sjeng* and *sphinx3* have significant improvements over the direct-mapped cache when the DAC design is used. The *DAC-Budget* outperforms *DAC-Deluxe* and direct-mapped cache, because it stays in set-associative mode for longer time, as shown in Figure 5.

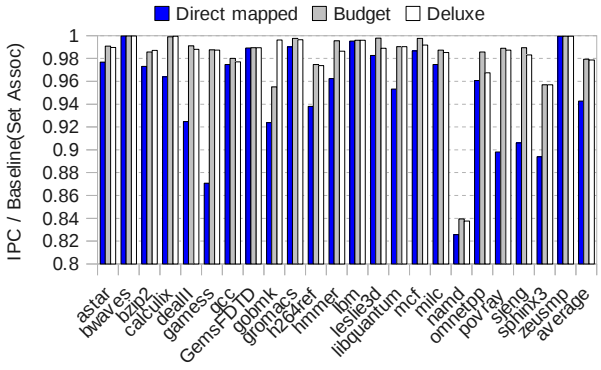

Fig. 7. IPC of DAC and Direct-Mapped Cache normalized to Set-Associative IPC

Finally, to estimate the impact of DAC on the cache delay, we incorporated the DAC switching feature into CACTI. The extra delay of DAC comes from the multiplexer to select the cache mode as direct-mapped or set-associative, and from the small decoder that uses the least significant bits of the tag to select the way to be activated in the direct-mapped mode. These two activities happen simultaneously, and the maximum delay of this mechanism is 0.013 ns, which is only 1.6% of the set-associative cache access time(0.798 ns).

VI. Related Work

A recent work [11] introduced the Tag-Less Cache design that is built around the concept of extended TLB (eTLB). In this scheme, the way information for every memory line in a page is stored within the extended TLB entry. The way information is then directly used for accessing only one cache way, thus achieving the same cache energy savings on every cache access as the DA cache proposed in this paper. However, the Tag-Less cache design incurs non-trivial design changes, as the TLBs have to be significantly expanded. For example, for 4KB pages and 64-byte cache lines, 64 way indexes have to be added to each TLB entry. In addition, the tag-less cache increases the delay of the cache access, because the cache index information has to be read out from the TLB before the selected way can be activated. This increase in the cache access time is larger than what is encountered in DAC, where a simple decoder directly reads the way bits from the virtual address and activates one of the ways based on that. In addition, the design of [11] requires additional logic (e.g. reverse TLB) to handle synonyms. In contrast, the DAC design handles synonyms with no additional complexity, as it is basically implemented within the framework of a virtually-indexed physically-tagged set-associative cache.

Some previous efforts improved the performance of direct-mapped and low-associativity caches by adding pseudo-associativity on top of these designs [12], [9] to provide a more flexible placement. This is implemented by providing different hashing functions for different ways or sets, and also

the capability to migrate the lines between the sets. Essentially, some over-utilized sets can spill into underutilized sets. These techniques significantly alter the core cache circuitry, and it is unclear if they are applicable at the level of L1 caches. In contrast, the DAC design only needs a simple control circuitry to switch between two traditional cache modes, and a simple way to implement direct-mapped style accesses within the framework of set-associative caches.

A technique to dynamically adjust the cache index bits to minimize conflict misses in a direct-mapped cache was proposed in [10]. The Balanced cache design [15] widens the decoder length and incorporates programmable address decoders to similarly achieve a more balanced distribution of accesses across the lines of a direct-mapped cache. In contrast to these techniques that improve the performance of direct-mapped cache, the DAC design simply switches to set-associative mode of operation after detecting such a need. This requires almost no changes to the core cache circuitry.

Another mechanism for reducing the energy expended in the activation of multiple cache ways is by way prediction [2], [7], [1]. The way prediction information for each set is kept in a separate table, and only the predicted way get activated. On a mis-prediction, additional cycle has to be spent checking the remaining ways, and also additional energy is consumed in this process. In contrast, DAC does not rely on speculation, because the information about the way where the data can be located is directly available from the least significant bits of the address tag. Another technique used in [7], [1] is selective direct-mapping. As in DAC, the position of some lines in the cache is directly determined by the address. However, since selective direct-mapping puts lines is direct-mapped positions selectively, per-line tracking logic is required by the scheme making it more complex than DAC. In contrast, DAC switches the entire cache either to direct-mapped or set-assoctaive regime using a simple algorithm.

Circuit-level techniques have been also proposed to reduce the cache energy. For example, [3] proposed multiple-line buffers, bit-line segmentation and sub-banking techniques to reduce the switching capacitance in the course of accessing the L1 cache. These techniques are orthogonal to DAC and can be utilized in conjunction with our design to further improve the cache energy-efficiency. While this paper addresses dynamic adaptation of caches, adaptatiin was also applied to other datapath resources [6].

VII. Concluding Remarks

In this paper we presented DAC - a design that dynamically changes cache associativity during the program execution in response to its cache demands. Where the programs can take advantage of high associativity, DAC control logic automatically switches the cache to the set-associative mode. However, in situations where no extra performance can be harvested as a result of increased associativity, DAC performs direct-mapped style accesses by only activating one of the cache ways. DAC does not significantly alter the core cache circuitry, nor does it rely on any speculation to determine which way to access. DAC only adds a simple control logic and a few extra tags to control the switching between the operating modes. Since DAC fundamentally operates within the framework of a traditional set-associative cache, all design intricacies, such as the detection of possible aliases in the cache can be easily handled using already existing mechanisms. Our results show that DAC is an effective technique that can significantly reduce the dynamic energy of the L1 data cache with minimal impact on complexity and performance. The proposed technique can also be applied to the instruction caches - this is left for the future work.

References

[1] B. Batson and T. Vijaykumar, "Reactive-associative caches," in *Proceedings of PACT*, 2001.

[2] K. Inoue, T. Ishihara, and K. Murakami, "Way-predicting set-associative cache for high performance and low energy consumption," in *Proceedings of ISLPED*, 1999.

[3] M. Kamble and K. Ghose, "Reducing power in superscalar processor caches using subbanking, multiple line buffers and bitline segmentation," in *Proceedings of ISLPED*, 1999.

[4] "M-sim version 3.0, code and documentation," 2005, available online at: http://www.cs.binghamton.edu/~msim.

[5] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," HP Laboratories, Tech. Rep. HPL-2009-85, April 2009, available online at: http://www.hpl.hp.com/techreports/2009/HPL-2009-85.html.

[6] D. Ponomarev, G. Kucuk, and K. Ghose, "Dynamic allocation of datapath resources for low power," in *Workshop on Complexity-Effective Design, held in conjunction with ISCA*, 2001.

[7] M. Powell, A. Agarwal, T. Vijaykumar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," in *Proceedings of MICRO*, 2001.

[8] M. Qureshi and Y. Patt, "Utility-based cache partitioning," in *Proceedings of MICRO*, 2006.

[9] M. Qureshi, D. Thompson, and Y. Patt, "The v-way cache: Demand-based associativity via global replacement," in *Proceedings of ISCA*, 2005.

[10] A. Ros, P. Xekalakis, M. Cintra, M. Acacio, and J. Garcia, "Ascib: Adaptive selection of cache indexing bits for removing conflict misses," in *Proceedings of ISLPED*, 2012.

[11] A. Sembrant, E. Hagersten, and D. Black-Shaffer, "Tlc: A tag-less cache for reducing dynamic first level cache energy," in *Proceedings of MICRO*, 2013.

[12] A. Seznec, "A case for two-way skewed-associative caches," in *Proceedings of ISCA*, 1993.

[13] A. Sodani, "Race to exascale: Opportunities and challenges. keynote talk." in *MICRO Symposium*, 2011.

[14] C. D. Spradling, "Spec cpu2006 benchmark tools," *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 130–134, 2007.

[15] C. Zhang, "Balanced cache: Reducing conflict misses in direct-mapped caches," in *Proceedings of ISCA*, 2006.

[16] H. Zhou, M. Toburen, E. Rotenberg, and T. Conte, "Adaptive mode control: A static-power-efficient cache design," in *Proceedings of PACT*, 2001.