Access-Mode Predictions for Low-Power Cache Design

AN ACCESS-MODE PREDICTION TECHNIQUE BASED ON CACHE HIT AND MISS SPECULATION FOR CACHE DESIGN ACHIEVES MINIMAL ENERGY CONSUMPTION. USING THIS METHOD, CACHE ACCESSES CAN BE ADAPTIVELY SWITCHED BETWEEN THE WAY-PREDICTION AND THE PHASED ACCESSING MODES.

Zhichun Zhu
Xiaodong Zhang
College of William and Mary

The successful pursuit of high performance on computer systems has produced the negative by-product of high power dissipation. Circuit-level techniques alone can no longer keep power dissipation under a reasonable level. Researchers have made efforts to reduce power dissipation at the architectural level by producing such schemes as reducing on-chip cache power consumption—a major power consumer in microprocessors.[1] For example, experts believe that the processor power consumed by on-chip caches will increase from the 15 percent in Compaq's Alpha 21264 to 26 percent in future Alpha 21464 processors.[2]

A set-associative cache is commonly used in modern computer systems for its ability to reduce cache conflict misses. However, a conventional set-associative cache implementation is not power-efficient. As Figure 1a shows, a conventional *n*-way set-associative cache probes all *n* blocks (both tag and data portions) in a set but, at most, will only really use one block. The percentage of wasted energy will increase as cache associativity *n* increases. High-associativity caches already exist in some commercial processors. For example, the Intel Pentium 4 processor exploits four-way L1 caches and an eight-way L2 cache.

An effective approach to reduce power consumption in set associative caches is lowering the number of memory cells involved in an access. One method divides each data RAM into multiple sub-banks and only activates words at the required offset from all cache ways.[3] Another alternative is to selectively disable a subset of cache ways during execution periods with modest cache activity,[4] or to dynamically resize both the number of sets and ways in caches.[5] The phased cache first compares all the tags with the accessing address, then probes only the desired data way.[6] Way-prediction is another effective approach that speculatively selects a way to access before making a normal cache access.

Figures 1b and 1c illustrate the access patterns for phased and way-prediction *n*-way set-associative caches. Compared with the conventional implementation, the phased cache only probes one data subarray instead of *n* data subarrays (each way comprises a tag subarray and a data subarray). However, the sequential accesses of tag and data will increase the cache access latency. The way-prediction cache first accesses the tag and data subarrays of the predicted way. If the prediction is not correct, it then probes the rest of tag and data subarrays simultaneously. An access in a phased cache consumes more energy and has longer latency than a correctly predicted access in way-prediction cache, but consumes less energy than a mispredicted access. Hence, when the prediction accuracy is high, the way-prediction cache is more energy-efficient than the phased cache.[7]

0272-1732/02/$17.00 © 2002 IEEE

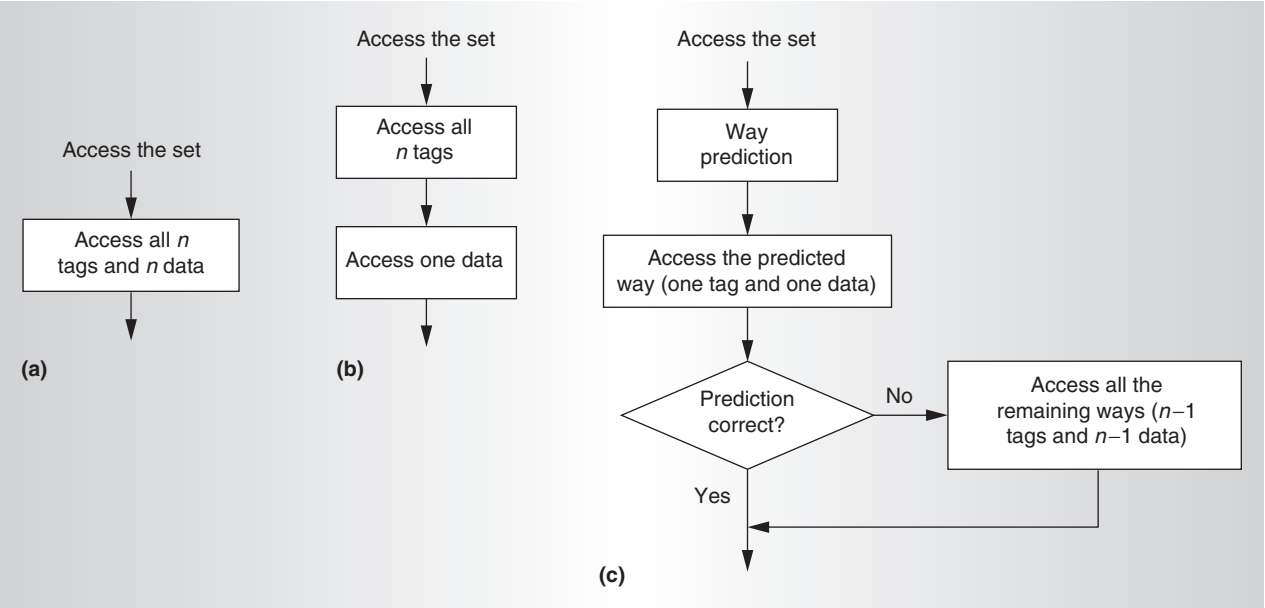

Figure 1. Access patterns for a conventional *n*-way set-associative cache (a), a phased *n*-way set-associative cache (b), and a way-prediction *n*-way set-associative cache (c).

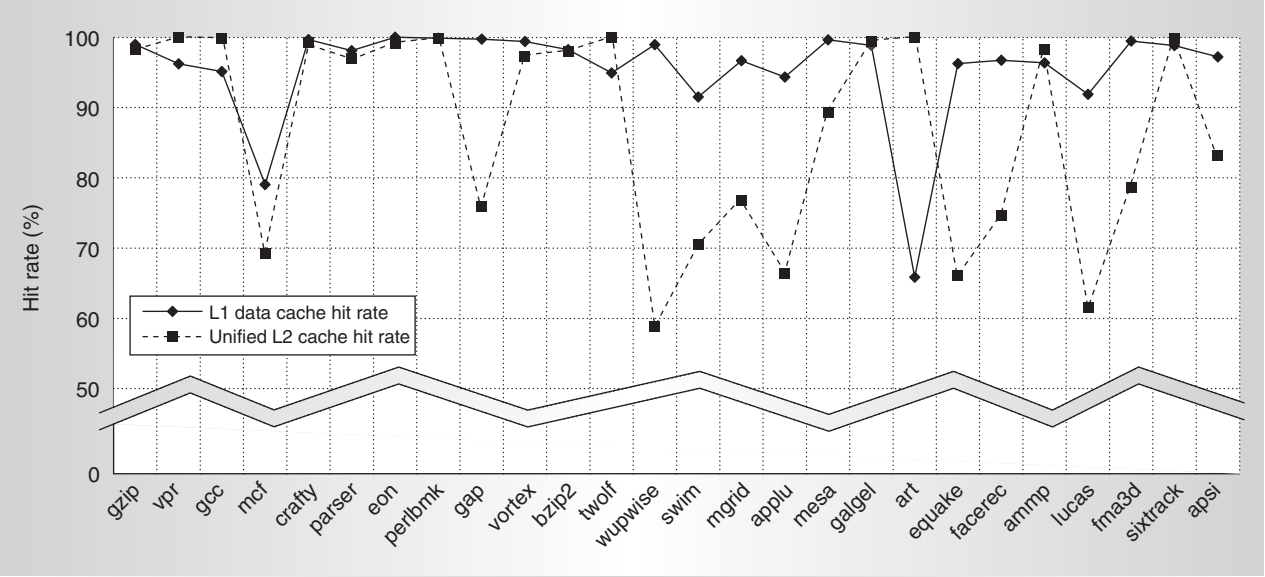

Figure 2. Hit rates for SPEC2000 benchmarks on an eight-way 64-Kbyte data cache with a 64-byte block size and an eight-way 4-Mbyte unified L2 cache with a 128-byte block size.

The way-prediction hit rate is bounded by the cache hit rate, which is highly application dependent. Figure 2 shows the hit rates for all the SPEC2000 benchmark programs on a 64-Kbyte data cache and a 4-Mbyte L2 cache. Most applications have L1 data cache hit rates as high as 95 percent with a few exceptions. (The art benchmark is one exception; its hit rate is only 65 percent.) The distribution of L2 cache hit rates is even more diverse than that of L1 cache hit rates. Fourteen of the 26 programs have L2 cache hit rates higher than 95 percent; five programs have L2 cache hit rates below 70 percent.

Based on these observations, we can minimize the energy consumption of a cache access

by using the way-prediction mode to handle a cache hit and by using the phased mode to handle a cache miss. Motivated by this relationship between a cache access mode and its energy consumption, and by the application dependent cache-hit patterns, we propose an access-mode prediction technique based on cache hit/miss prediction.[8,9] Our approach for low-power cache design combines the energy-efficient merits of both phased and way-prediction cache structures. In the case of predicted cache hits, the way-prediction scheme determines the desired way and probes that way only. In the case of predicted misses, the phased scheme accesses all tags first, then probes only the appropriate way. We call this an access-mode-prediction (AMP) cache.

Since we target low-power cache design, the access-mode predictor should only add a small amount of overhead on both latency and energy consumption. We derive a simple predictor from existing branch prediction techniques; this predictor uses a global-access history register and a global pattern history table to obtain high prediction accuracy.

We also optimize the way-prediction technique for energy reduction purposes. We find that the multicolumn-based way-prediction is highly effective for energy reduction in high-associativity caches. Previous studies have shown that way-prediction based on most-recently-used (MRU) policy can effectively reduce the energy consumption of set-associative caches. Our experimental results indicate that for four-, eight-, and sixteen-way caches, multicolumn-based way-prediction can reduce the energy consumption by 7 percent, 17 percent, and 40 percent on average, compared with MRU-based way-prediction. In addition, the multicolumn-based way-prediction is a nearly optimal technique, which can correctly predict the locations of 98 percent of cache hits on average. Compared with the already optimized four-way L1 and eight-way L2 multicolumn caches, the access-mode prediction technique can further reduce the energy consumption and energy-delay product by 9 percent on average (up to 46 percent). Compared with the four-way L1 and eight-way L2 phased caches, this prediction technique can also reduce the energy consumption by 20 percent on average (up to 27 percent) and reduce the energy-delay product by 25 percent on average (up to 35 percent).

Experimental environment

We use Cacti 2.0 to estimate the timing and power consumption of different implementations of set-associative caches.[10] For AMP caches, we estimate the energy consumption and access latency of first hits, non-first hits, misses, and write backs in both way-prediction and phased-access modes. The estimation methodology we used is similar to that described by Powell et al.[11] We assume that the AMP, multicolumn, and phased caches all have the same array organization as the conventional set-associative cache. To estimate the energy consumption of each cache structure, we assume that only the part of a cache being accessed consumes energy. For example, for a first-hit load, one tag subarray, one data subarray, and the output driver consume the energy. The access-latency estimation depends on whether the tag and data portion are accessed concurrently. For example, the access latency for a first-hit load is equivalent to the maximum value between the tag subarray access time and the data subarray access time, plus the data output times.

We use the SimpleScalar tool to collect the program execution statistics.[12] We modified the cache part of the simulator to evaluate the behavior of different way-prediction and mode-prediction approaches.

The simulated 1-GHz eight-issue processor has separate 64-Kbyte instruction and data caches with block sizes of 64 bytes, and a unified 4-Mbyte L2 cache with a block size of 128 bytes. This configuration is similar to those used in high-end workstations, such as the Alpha 21264 and Sun Microsystems' UltraSparc III. The associativity of L1 and L2 caches ranges from four to 16. L1 and L2 caches have four and two sub-banks in each data RAM, respectively.[3] We assume the use of 0.18-micron technology and use the precompiled Alpha version of SPEC2000 binaries as the workload. Our experiments used the reference input data files, and we fast-forward the first four billion instructions, then collect detailed statistics on the next one billion instructions.

Access-mode predictions

The effectiveness of our scheme mainly depends on the accuracy of predicting cache hits or misses.

Strategy and energy consumption

Our motivation for using access-mode predictions in low-power cache design comes from the observation that neither way-prediction nor phased caches are energy efficient for both cache misses and hits. We use a simplified timing and energy model to quantify this observation. For the different types of caches in our experiments, we estimated the latency and energy consumption based on the Cacti timing and power consumption model.[10]

Let E_{tag} and E_{data} be the energy consumed by a tag subarray and a data subarray upon a reference. For a correctly predicted hit in the way-prediction cache, the energy consumed is $E_{tag} + E_{data}$, compared with $n \times E_{tag} + E_{data}$ in the phased cache, where n is the cache's associativity. On the other hand, a miss in the way-prediction cache will consume $(n + 1) \times E_{tag} + (n + 1) \times E_{data}$, in comparison with $(n + 1) \times E_{tag} + E_{data}$ in the phased cache. Regarding the access latency, a correctly predicted hit in the way-prediction cache takes one time unit, compared with two time units in the phased cache. Our objective is to pursue the lowest possible energy consumption and latency for both cache hits and misses.

The predictive phased cache first probes all the tag subarrays and the predicted data subarray.[13] This approach reduces the energy consumption of nonfirst hits in the way-prediction cache at the price of increasing energy consumption for both first hits and cache misses. Powell et al. proposed a scheme that exploits direct-mapping accesses for the predicted nonconflicting accesses and way prediction for those predicted conflicting accesses.[11] This approach reduces the energy consumption for cache hits but does not optimize the consumption for cache misses.

Figure 3 presents the access sequence controlled by access-mode predictions. Upon a reference, the access-mode predictor makes a decision based on the cache access history. If the predictor indicates that the way-prediction mode should be used, the way-prediction scheme will handle the following access sequence. Otherwise, the phased-access scheme will handle the sequence.

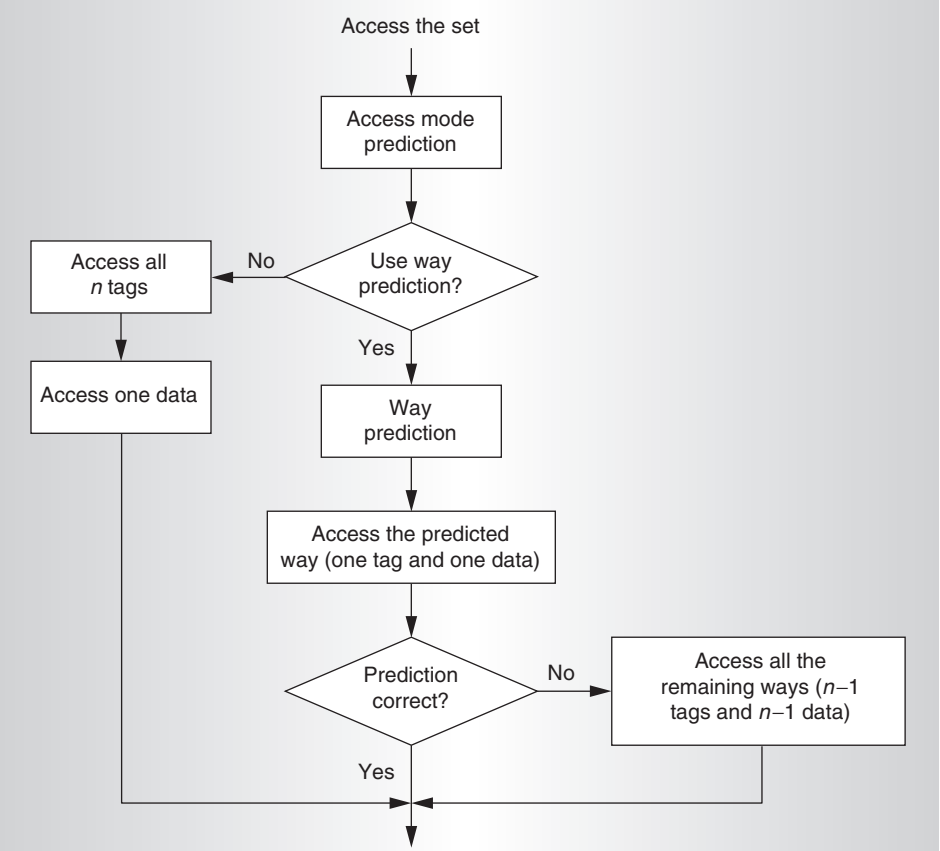

Figure 3. Access pattern of n-way access-mode prediction cache.

For an AMP cache with perfect access-mode prediction and way prediction, the energy consumption is the lower bound of energy consumption for the set-associative cache. The accuracy of access-mode and way-prediction predictors determines misprediction overhead.

Predictors

Yoaz et al. propose using cache hit/miss prediction to improve load instruction scheduling.[9] To reduce the memory bandwidth requirement, Tyson et al. use miss prediction to dynamically mark which load instruction is cacheable and which is nonallocatable.[8]

Technically, nearly all branch prediction tech-

Table 1. Comparison of misprediction rates for access-mode predictors. The separate 64-Kbyte instruction cache and data cache are eight-way with a 64-byte block size. The unified L2 cache is eight-way with a 128-byte block size.

				Misprediction rate (%)		
Access-mode predictor	**L1 instruction cache (Bytes)**	**L1 data cache (Bytes)**	**Unified L2 cache (Kbytes)**	**L1 instruction cache**	**L1 data cache**	**Unified L2 cache**
Saturating counter	256	256	8	0.11	5.68	14.44
GAg	~256	~256	~8	0.12	4.97	5.51
PAg	~1K	~1K	~56	0.13	3.94	3.83
(2, 2)	~1K	~1K	~32	0.11	5.27	13.43
gshare	~256	~256	~8	0.12	6.01	15.57

niques might be adapted. However, to reduce cache power consumption, the access-mode predictor must be simple. Thus we only present variants with low resource requirements.

Saturating counter. This prediction has the simplest implementation based on the two-bit saturating up/down counter.[14] Each set associates with a two-bit counter that is incremented for each hit and decremented for each miss. Each cache (instruction, data, or L2) has its own access-mode predictor; its reference address acts as the index to the prediction table.

Two-level adaptive predictor. Another alternative technique comes from the two-level adaptive branch predictor.[15] We implement both global adaptive two-level branch prediction using a global pattern-history table (GAg) and per-address two-level adaptive branch prediction using a global pattern-history table (PAg). In the GAg-derived implementation, a global k-bit access-history register records the results of the most recent k accesses. If the access is a hit, the register records a 1; otherwise, it records a 0. The global access-history register is the index to a global pattern-history table, which contains 2^k entries. Each entry is a two-bit saturating counter. In the PAg-based implementation, each set has its own access history register. All access-history registers index a single pattern history table. In our experiments, we set the number of entries in the global pattern-history table for both GAg and PAg predictors to the number of sets in the corresponding cache.

(M, N) correlation predictor. In this scheme, based on the approach proposed by Pan, So, and Rahmeh, an M-bit shift register stores the hit/miss history for the most recent M accesses.[16] Each set has 2^M entries, which the M-bit register indexes. Each entry is an N-bit counter. We apply a (2, 2) predictor in our experiments.

gshare predictor. McFarling originally proposed the global share (gshare) predictor.[17] The exclusive-OR of the global access-history with the current reference's set number provides the indexes for the global pattern-history table. The number of table entries equals the number of sets in the cache for our experiments.

Accuracy

Table 1 compares the misprediction rates of the five policies we consider here on a system with eight-way 64-Kbyte instruction/data caches and a 4-Mbyte L2 cache. The table lists values averaged over all 26 SPEC2000 programs. Two-level adaptive predictor PAg obtains the lowest misprediction rates—below 4 percent on instruction, data, and L2 caches. However, the PAg version also occupies more area than other predictors. The GAg predictor has the second lowest misprediction rates—below 6 percent on both L1 and L2 caches—and requires the smallest additional area.

Overhead

We use the GAg predictor in our remaining experiments for several reasons. The prediction accuracy of GAg is only slightly lower than that of PAg. However, the GAg predictor requires much less additional chip area for recording access history than the PAg predictor. The area overhead of GAg predictors for instruction and data caches is only 0.05 percent, and only 0.02 percent for the L2 cache.

More importantly, unlike other predictors, the GAg predictor does not require the next cache reference's address to perform mode prediction. Thus, when a cache reference comes, the GAg predictor has already determined which access mode to perform, a decision based on the access history. The timing overhead introduced by the GAg predictor is trivial. Regarding energy consumption, by using mode predictors, our simulation results indicate that the overall overhead of instruction, data, and L2 caches is under 0.8 percent for all the benchmark programs.

Multicolumn-based way prediction

The effectiveness of the AMP cache on energy reduction depends not only on access-mode-predictor accuracy, but also on the underlying way-prediction technique.

Multicolumn cache

Researchers originally proposed way-prediction caches for reducing the average access latency of set-associative caches.[18] Some prediction strategies can predict the way containing the desired data and read data from that way first. Way-prediction techniques can provide power savings in set-associative caches.[7] If the first way-prediction is correct—we call this a first hit—a cache read only probes the desired way. In this case, the power consumption is close to that in a direct-mapped cache.

However, if the prediction is not correct, the way-prediction cache will consume more energy than a conventional implementation because of the additional operations to maintain the prediction mechanism. Thus, the accuracy of the prediction technique, measured by the first-hit rate, is crucial to reducing power consumption.

Previous studies have shown that an MRU-based way-prediction cache is more power efficient than other techniques.[7,19] (The MRU cache maintains an MRU list that marks the most recently used—or accessed—block for each set. For any reference mapping into the set, the search always begins from the MRU location.) For low-associativity caches, such as two-way or four-way caches, the MRU technique works well in predicting the desired way. However, as the cache associativity increases, the MRU structure might potentially decrease the first-hit rate. In an MRU cache, the number of search entries is equivalent to the number of sets. Doubling the cache associativity for a fixed cache size and a fixed block size halves the number of search entries in the MRU cache. Normally, this causes a decrease in way-prediction accuracy. For example, our experiments used the SPECint2000 program vpr. As the associativity of a 64-Kbyte, L1, MRU data cache increases from four to eight and then to 16, this cache's first-hit rate drops from 90.3 to 85.4, then to 80.8 percent.

The multicolumn cache addresses this MRU-cache limitation.[18] A *major location* is the location on which a reference can be directly mapped. We use major location to guide way prediction. For an n-way set associative cache, a reference's major location is determined by the low-order, or the least significant, log n bits of its tag.

As the associativity increases, the first-hit rate does not change much in the multicolumn cache. The number of search entries in the multicolumn cache equals the product of the number of sets and the cache associativity. Thus, as the associativity increases for fixed cache and block sizes, the number of search entries in the multicolumn cache remains the same. Again using vpr as an example, as the associativity of a 64-Kbyte L1 multicolumn data cache increases from four to eight, then to 16, the first-hit rate remains the same (93.4 percent).

Power considerations

In the original design of multicolumn caches, a swapping mechanism ensures that the MRU block always resides at the major location after a reference (though other blocks might replace it later). From a power consumption point of view, swapping is an expensive operation because two cache ways are involved in each swapping operation. So the power consumption of swapping approximately equals the cost of two accesses to a single way.

To eliminate swapping, we propose a power-efficient variation for the multicolumn cache. The cache maintains an index entry for each major location in a set to record its MRU information.

Figures 4a and 4b (next page) show multicolumn-cache implementations with and without swapping, and the access sequences for two references on a four-way, set-associa-

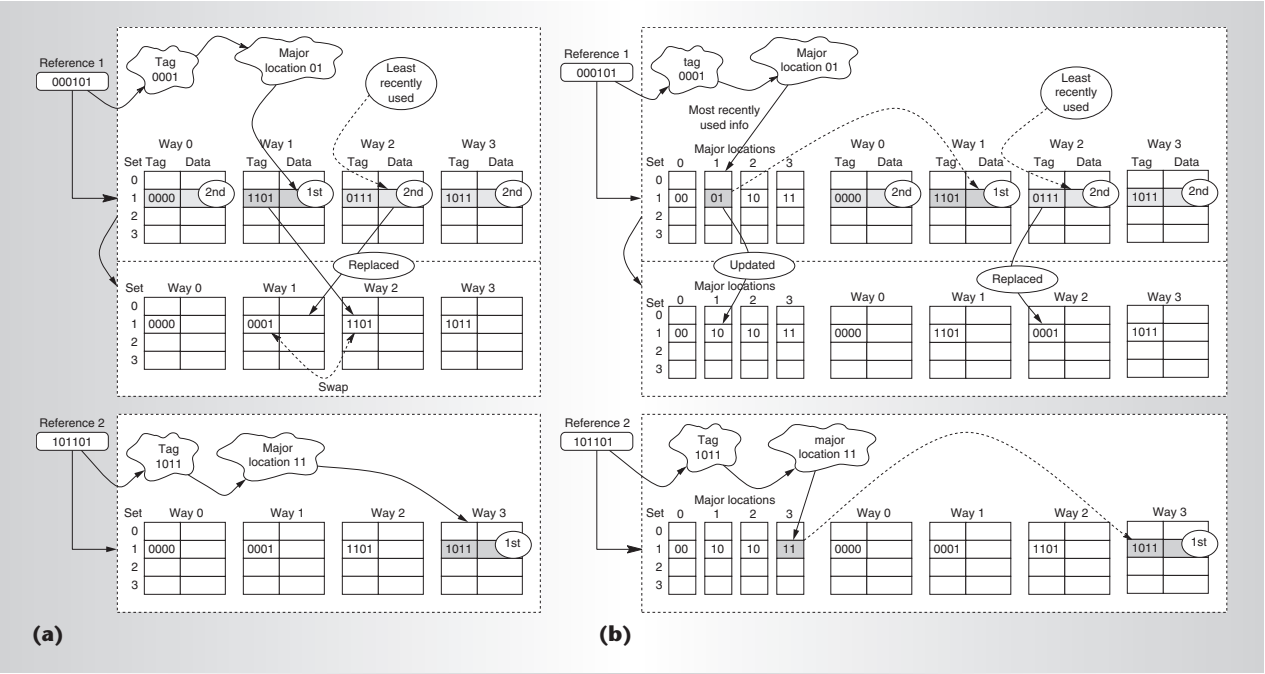

Figure 4. Way-prediction strategies for a multicolumn cache with (a) and without (b) swapping.

tive cache with four sets. For a reference, we only present tag and set portions. Thus, the first reference (000101) maps to set (01) with tag (0001), and its major location is way 1. The multicolumn cache with swapping begins a search at the major location. The multicolumn cache without swapping first retrieves the major location's MRU information, which points to way 1 in the figure, then begins the search from way 1. In either case in the figure, the predicted way does not contain the desired block, so the cache probes all remaining ways on the second attempt. This reference is a cache miss; the least-recently-used (LRU) block at way 2 is replaced. The swapping implementation places the new block (000101) at its major-location way 1, and swaps the block originally at way 1 (110101) to way 2. The implementation without swapping updates the MRU information of major location 1 by pointing to way 2, where the new block resides. By recording MRU information for each major location, the MRU block need not always reside at its major location, avoiding the energy-consuming swapping operations. The second reference (101101) is a first hit in both cases; hence, requiring no swappings and leaving the MRU information unchanged.

Tradeoffs

There are several performance and energy tradeoffs in the implementations of multicolumn caches. For the original design, the search always begins from a reference's major location, which is determined by a simple bit selection on the reference address. Thus, determining a search order neither lengthens cache access time nor consumes additional energy. However, maintaining this way-prediction mechanism occasionally requires the swapping operation, which consumes energy and can delay subsequent references.

On the other hand, the implementation without swapping avoids the energy-consuming operation but uses additional cache area for recording MRU information. It also consumes additional energy for retrieving MRU information and could increase cache access time. Since this implementation uses the reference address to index the MRU information table, MRU information retrieval might lengthen cache access time. However, we can apply the same arguments for using an MRU cache to using a multicolumn cache. If the reference address is available earlier, the cache access could begin at an earlier pipeline stage. In addition, overhead for fetching MRU information could be tolerated for L2 caches.

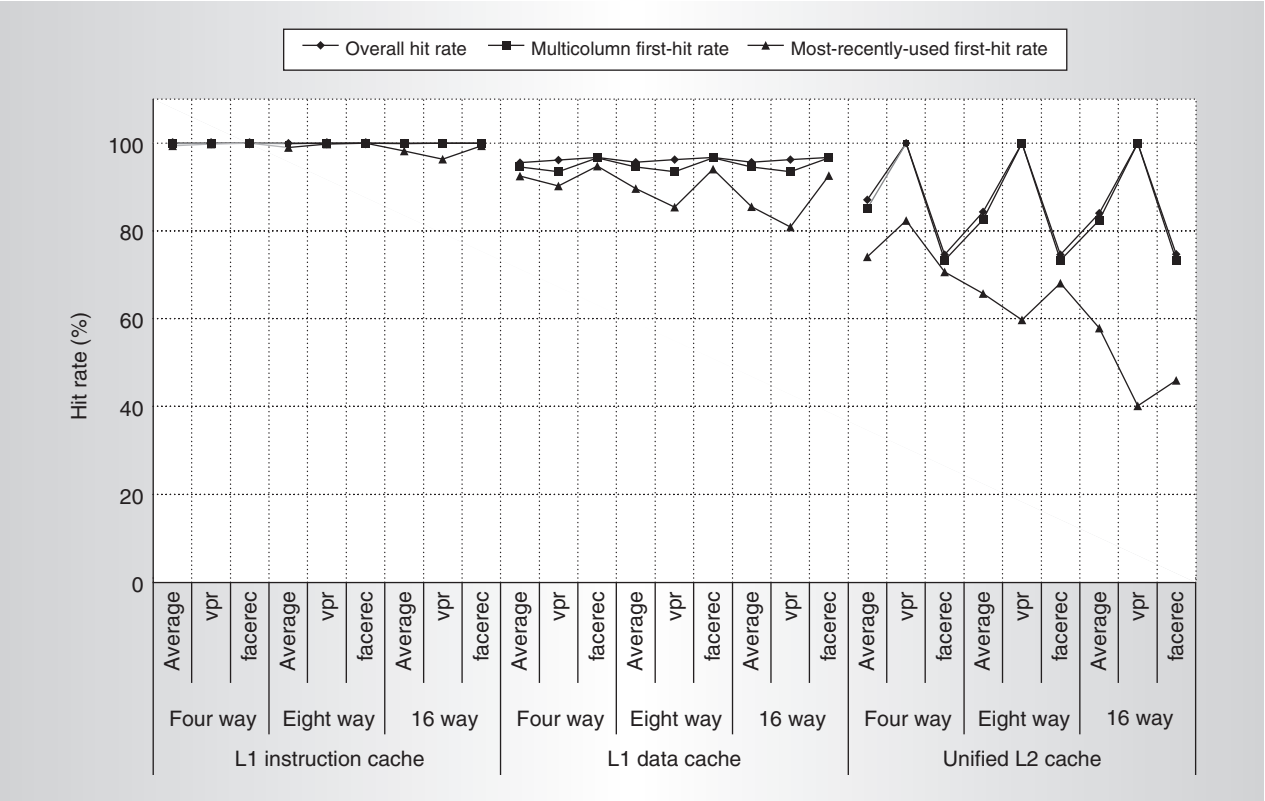

Figure 5. First-hit rates for multicolumn and most-recently-used caches.

Regarding the area, the overhead of recording the MRU information is log (associativity) ÷ (block size × 8). Because cache block size in a modern computer system is normally at least 32 bytes, this overhead is trivial. The energy consumption overhead for accessing the small MRU table is also trivial.

We evaluate the frequency of swapping operations in eight-way multicolumn caches under our default system configuration. For all the SPEC2000 benchmarks, on average, only 0.2 percent of references to the instruction cache involve swapping operations, while 3.8 and 12 percent of references to the data and L2 caches involve them. A reference to an L1 instruction cache has a low probability of involving a swapping operation. This makes it more feasible to directly use major location to guide the search in the latency-sensitive L1 cache, despite the need for swapping operations.

On the other hand, the possibility that a reference to the L2 cache will require swapping is as high as 12 percent, which would consume about 20 percent more energy than in the implementation without swapping. In this case, it is more feasible to apply the multicolumn implementation without swapping and overlap the fetch of MRU information with the accesses to L1 caches.

Experimental results

In our experiments, we use the multicolumn cache with swapping for instruction and data caches, which are latency-sensitive and prone to relatively few swapping operations. We apply the implementation without swapping for the less latency-sensitive L2 cache because references to it will likely involve many swapping operations.

Multicolumn and MRU cache comparison

As stated previously, the first-hit rate is crucial to reducing power consumption in way-prediction caches. Figure 5 shows the first-hit rates for multicolumn and MRU caches as cache associativity increases. Due to space limitations, we only present the average first-hit rates of all the SPEC2000 programs and the first-hit rates of two representative benchmarks: vpr (an integer program) and facerec (a floating-point pro-

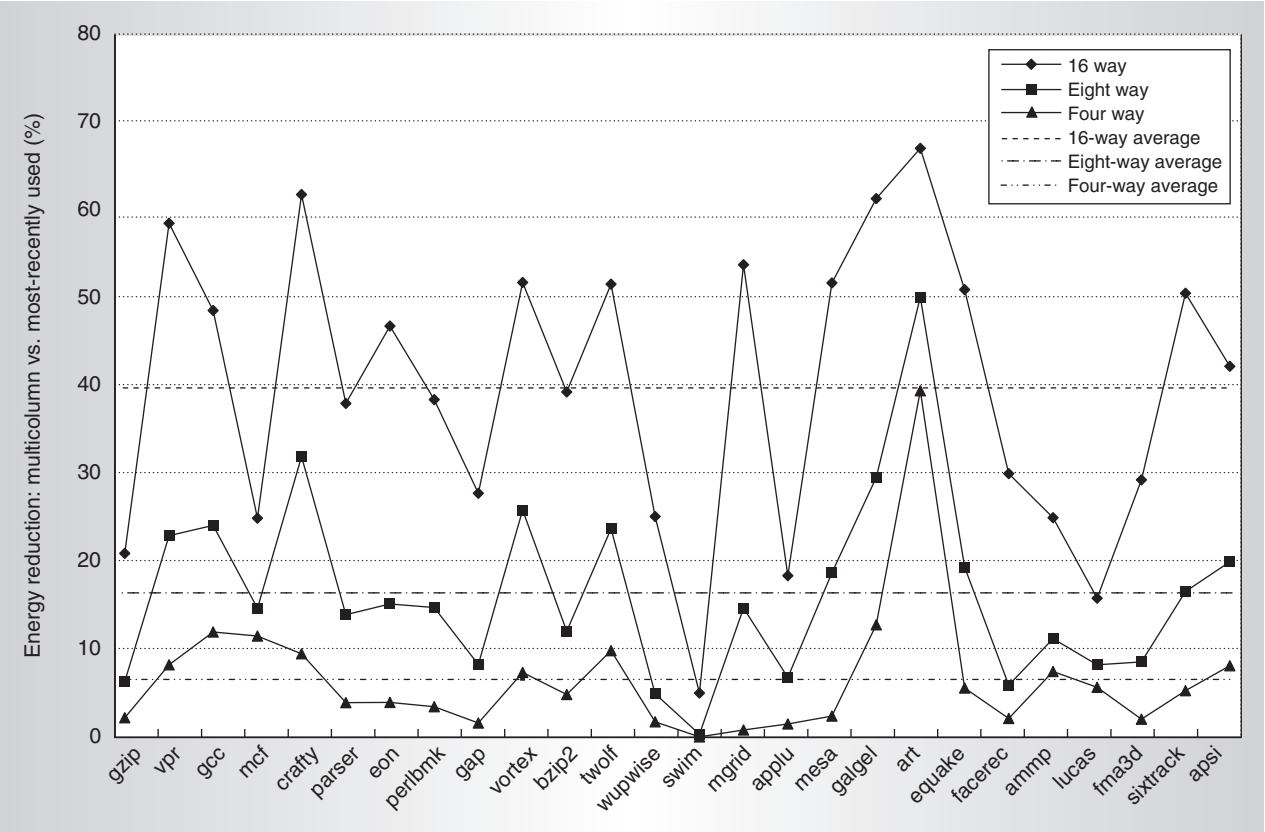

Figure 6. Energy reduction of multicolumn caches compared with most-recently-used caches.

gram). First-hit rates for MRU caches decrease as the cache associativity increases. In contrast, the first-hit rates of multicolumn caches remain almost unchanged. The first-hit rates of multicolumn caches are very close to the overall cache hit rates—which are the upper bounds of first-hit rates—at different cache levels and for different cache organizations. In all cases, the average first-hit rates for multicolumn caches are at least 98 percent of the average overall cache-hit rates. On the other hand, the first-hit rates for MRU caches are noticeably different from the overall cache-hit rates. For example, the 16-way L2 cache's MRU-based way-prediction has a first-hit rate that is only 69 percent of the average overall hit rate.

In terms of first-hit rates, the multicolumn-based way-prediction is a nearly optimal way-prediction technique. However, high first-hit rate is not our final target. Figure 6 illustrates the effectiveness of multicolumn-based way-prediction in reducing cache energy consumption. Compared with MRU-based way-prediction, the multicolumn-based technique can reduce the overall energy consumption of four-way instruction, data, and L2 caches by 0.1 percent to 39.6 percent (6.8 percent on average). As cache associativity increases to eight, the energy reduction is 0.4 percent to 50.3 percent (16.6 percent on average). For 16-way caches, the energy reduction of multicolumn caches is even more promising, with a reduction of 5.1 percent to 67.3 percent (40.0 percent on average).

AMP-cache energy reduction

We considered energy reduction across all the caches in the implementation (instruction, data, and L2), and then also looked at only data and L2 caches. This is useful because phased instruction cache consistently consumes more energy than both multicolumn and AMP instruction caches. By studying just data and L2 caches alone, we can make fair comparisons between different schemes.

Instruction, data, and L2 caches. Figure 7 compares the energy consumption of multicol-

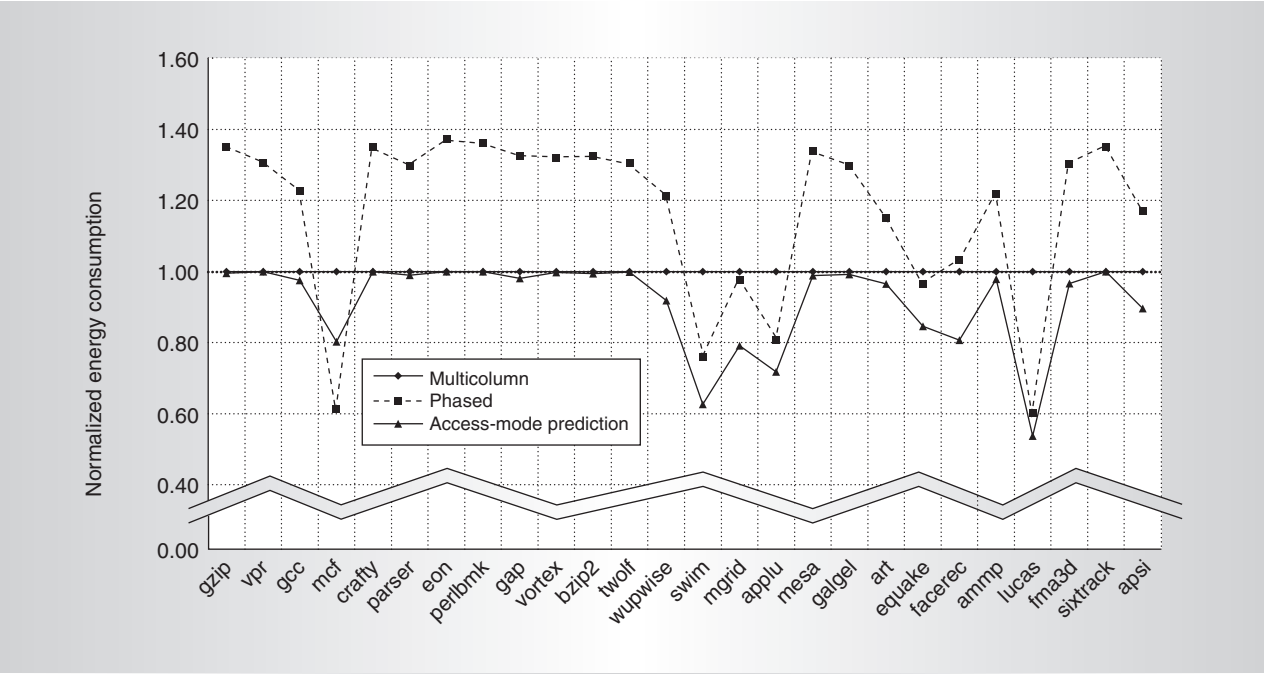

Figure 7. Energy consumption of multicolumn, phased, and AMP caches. The system has four-way 64-Kbyte instruction and data caches, and an eight-way 4-Mbyte L2 cache.

umn, phased, and AMP caches. (The AMP cache uses the GAg-based access-mode prediction and the multicolumn-based way-prediction). We have measured the energy consumed by four-way instruction and data caches, and eight-way L2 caches. We normalized all the values with respect to the energy consumed by the multicolumn caches.

The results show that the relative energy consumption of the multicolumn and phased caches is application-dependent. From among the 26 benchmark programs, phased caches consume less energy than multicolumn caches for six programs. Compared with multicolumn caches, phased caches might consume up to 37 percent more energy or up to 40 percent less energy, depending on the program behavior. On average, phased caches consume 16.7 percent more energy than multicolumn caches.

For all the programs, the AMP caches consume less energy than multicolumn caches. The energy reduction is 8.6 percent on average and up to 46.2 percent. Compared with phased caches, the AMP caches consume less energy for all the programs but one (mcf). This exception is because of the relatively high misprediction rates on data and L2 caches for this program. For mcf, the AMP caches consume 31.1 percent more energy than phased caches. For other programs, the access-mode prediction technique can reduce the energy consumption of phased caches by 10.6 percent to 27.1 percent. For all the programs, the average energy reduction of AMP caches is 20 percent, compared with phased caches.

Further looking into the decomposition of energy consumed by instruction, data, and L2 caches, we find that phased instruction caches consistently consume more energy than both multicolumn and AMP instruction caches. This is because of the high first-hit rates of multicolumn and AMP instruction caches. Compared with these two caches, the phased cache consumes more energy for first hits. The percentage of energy consumed by phased instruction caches is application dependent, ranging from 26.4 percent (for art) to 82 percent (for fma3d).

Data and L2 caches alone. Figure 8 (next page) compares energy consumption per instruction for only data and L2 caches, across the three different cache implementations. For 17 of the 26 programs, the multicolumn data and L2 caches consume less energy—up to 26.6 percent less—than the phased data and L2 caches. For the other nine programs, the

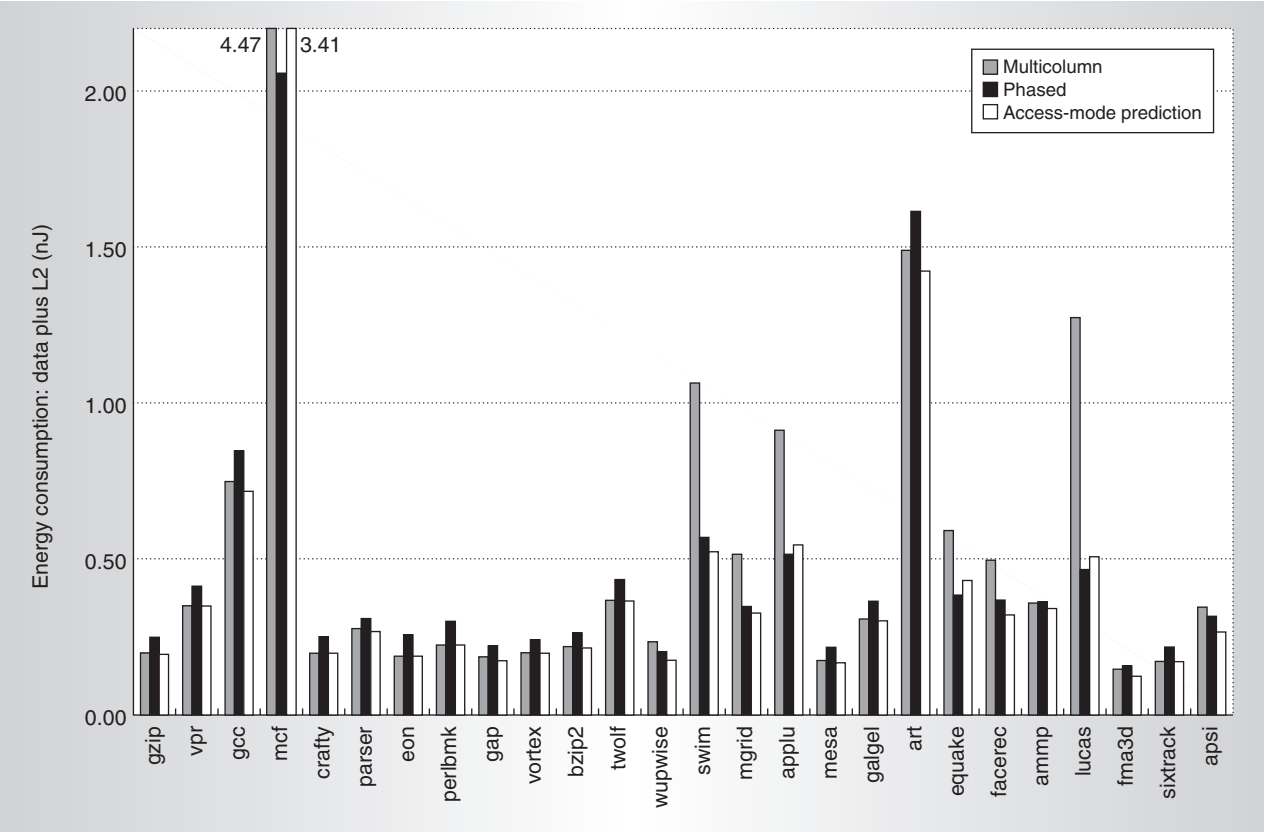

Figure 8. Energy consumed by data and L2 caches.

phased implementation consumes less energy—by up to 63.4 percent less—than the multicolumn implementation. On average for all 26 benchmark programs, the energy consumption of multicolumn and phased implementations was comparable.

For all 26 programs, data and L2 caches based on AMP consume less energy than those based on the multicolumn technique by up to 60.2 percent; the average is 14.4 percent. For 22 programs, the AMP caches reduced energy consumption compared to the phased implementation by 6.1 percent to 26.7 percent. For the remaining four programs, the phased implementation consumes less energy than the AMP caches by 5.5 percent to 39.7 percent. On average, the data and L2 caches based on AMP consumed 10.7 percent less energy than those phased caches.

The multicolumn cache consumes less energy for applications with high hit rates, while the phased cache consumes less energy for applications with low hit rates. Hit rate is highly application-dependent, especially for the data and L2 caches. So neither the multicolumn cache nor the phased cache worked well for saving energy across a wide range of applications. In contrast, the AMP cache always consumes the lowest possible energy for both hits and misses. It consumes less energy than both the multicolumn and the phased caches for a wide range of applications.

AMP caches energy efficiency

Regarding access latency, a correctly predicted hit in the multicolumn cache or in the AMP cache has shorter latency than in a phased cache. Table 2 presents the access latencies for a way-prediction hit, a way-prediction miss, and a phased access; we estimated these latencies by using the Cacti model. Using these values, we set the cache access latency in our experiments as follows. For instruction and data caches, a way-prediction hit takes one cycle; a way-prediction miss or a phased access takes two cycles. For the L2 cache, a way-prediction hit takes six cycles; a way-prediction miss or a phased

access takes 12 cycles.

We were concerned that AMP caches might not efficiently handle access sequences with varied latencies because varied latencies might cause access contention to caches. Instead, we found that way-prediction caches, such as MRU caches, also had challenges in handling variable access latencies. Thus, in dealing with variable access latencies, AMP caches don't introduce any more complexity than do way-prediction caches.

This reasoning also extends to solving conflicts upon accessing the data array. The mixture of phased accesses with way-prediction hits does not introduce any more complexity than the mixture of way-prediction misses with way-prediction hits.

Figure 9 compares the performance and energy-delay product of the AMP cache with that of the multicolumn and phased caches.

Compared with the multicolumn cache, the AMP cache might mispredict some way-prediction hits and perform phased accesses. This only slightly increases the average cache-access latency and degrades the overall performance. Among the 26 programs, the AMP and multicolumn caches have the same cycles per instruction for seven programs. The maximum CPI increase is 1 percent; the average CPI increase is only 0.1 percent. This indicates that the access-mode predictor only mispredicts a very small percentage of way-prediction hits as phased accesses. The AMP

Table 2. Access latencies for a way-prediction hit and miss, and phased access. These values are for 64-Kbyte four-way L1 and 4-Mbyte eight-way L2 caches.

	L1 instruction or data caches		L2 cache	
Access types	**Time (ns)**	**No. of cycles**	**Time (ns)**	**No. of cycles**
Way-prediction hit	0.90	1	5.85	6
Way-prediction miss	1.61	2	11.39	12
Phased access	1.88	2	10.77	12

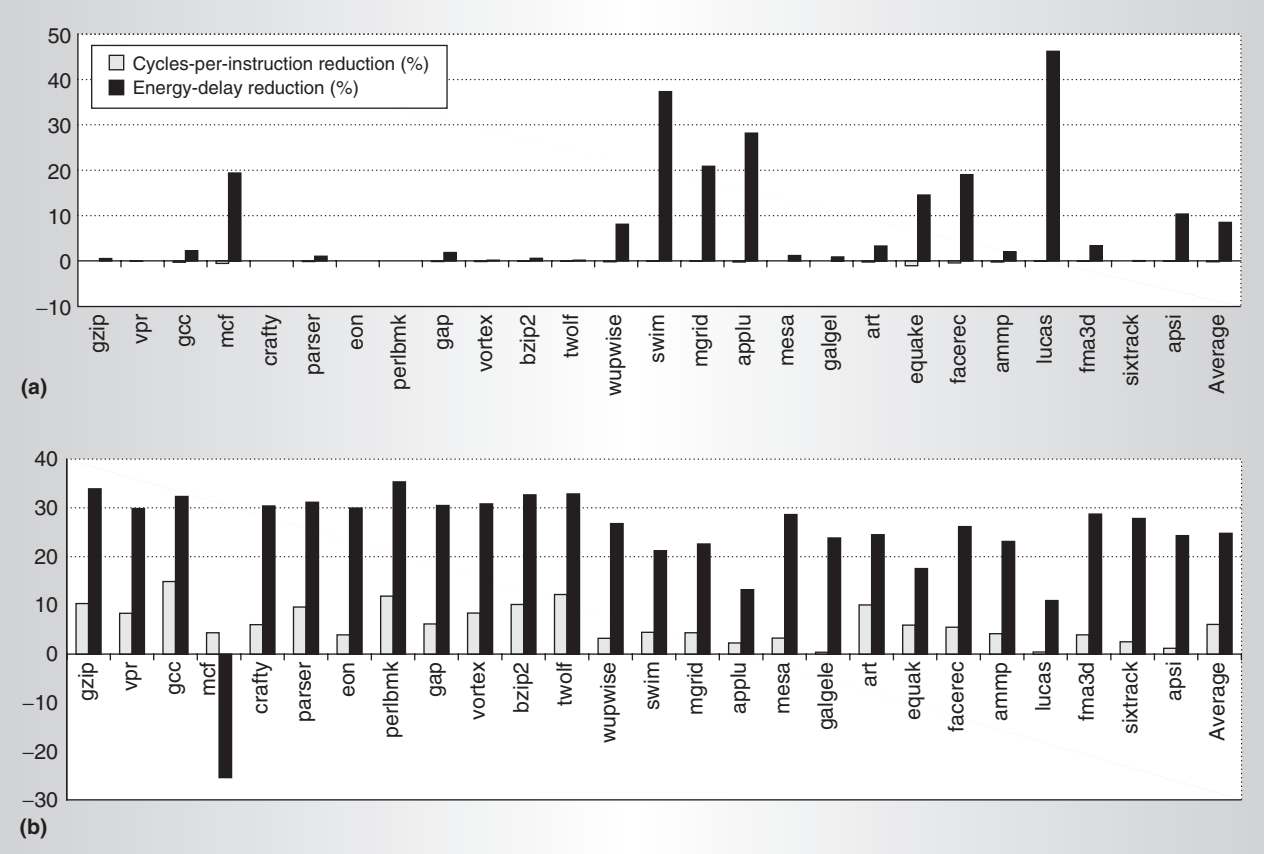

Figure 9. Reductions in cycles per instruction and energy-delay product for the AMP cache when compared with the multicolumn (a) and phased (b) caches.

cache obtains almost the same performance as the multicolumn cache.

Way-prediction hits have shorter latencies than those of phased accesses. Thus, the average access latency in the AMP cache is shorter than that of the phased cache. Compared with the phased cache, the CPI reduction of the AMP cache ranges from 0.4 to 14.9 percent and is 6.1 percent on average.

The AMP cache reduces the multicolumn cache's energy-delay product by 8.5 percent on average (up to 46.2 percent). The AMP cache's energy-delay product is 24.8 percent less on average (up to 35.3 percent) than that of the phased cache.

With a simple access-mode prediction based on cache hit and miss prediction, the AMP cache can effectively reduce energy consumption for a wide range of applications under systems with moderate- to high-associativity caches. The AMP cache can exploit the same mechanism used in the way-prediction cache to handle the varied latencies from different access modes and way-prediction hits and misses. The additional overhead of access-mode predictor and multicolumn-based way-prediction is negligible in terms of cache area, access latency, and energy consumption. We are using cache access information to adjust processor issue rates for low-power chip design. MICRO

Acknowledgment

We thank the anonymous referees for their constructive comments on our work. This work is part of an independent research project sponsored by the National Science Foundation for its program directors and visiting scientists.

References

1. D.M. Brooks et al., "Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors, *IEEE Micro*, vol. 20, no. 6, Nov.-Dec. 2000, pp. 26-44.
2. K. Wilcox and S. Manne, "Alpha Processors: A History of Power Issues and a Look to the Future," *Cool Chips Tutorial: An Industrial Perspective on Low Power Processor Design, 32nd Ann. Int'l Symp. Microarchitecture*, 1999; http://huron.cs.ucdavis.edu/micro32/presentations/cool_chips.pdf.
3. C.L. Su and A.M. Despain, "Cache Design Trade-offs for Power and Performance Optimization: A Case Study," *Proc. 1995 Int'l Symp. Low Power Design*, IEEE CS Press, Los Alamitos, Calif., 1995. pp. 63-68.
4. D.H. Albonesi, "Selective Cache Ways: On-Demand Cache Resource Allocation, *Proc. 32nd Ann. ACM/IEEE Int'l Symp. Microarchitecture* (Micro-32) IEEE CS Press, Los Alamitos, Calif., 1999, pp. 248-259.
5. S.H. Yang et al., "Exploiting Choices in Resizable Cache Design to Optimize Deep-Submicron Processor Energy-Delay," *Proc. 8th Int'l Symp. High-Performance Computer Architecture* (HPCA 01), IEEE CS Press, Los Alamitos, Calif., 2001, pp. 151-161.
6. A. Hasegawa et al., "SH3: High Code Density, Low-Power," *IEEE Micro*, vol. 15, no. 6, Dec. 1995, pp. 11-19.
7. K. Inoue, T. Ishihara, and K. Murakami, "Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption," *Proc. Int'l Symp. Low Power Electronics and Design*, IEEE CS Press, Los Alamitos, Calif., 1999. .pp. 273-275.
8. G. Tyson et al. "A Modified Approach to Data Cache Management," *Proc. 28th Ann. Int'l Symp. Microarchitecture* (Micro-28), IEEE CS Press, Los Alamitos, Calif., 1995, pp. 93-103.
9. A. Yoaz et al., "Speculation Techniques for Improving Load Related Instruction Scheduling," *Proc. 26th Ann. Int'l Symp. Computer Architecture* (ISCA 99), IEEE CS Press, Los Alamitos, Calif., 1999, pp. 42-53.
10. G. Reinman and N. Jouppi, *An Integrated Cache Timing and Power Model*, tech. report, Compaq Western Research Lab, Palo Alto, Calif., 1999.
11. M.D. Powell et al., "Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct-Mapping," *Proc. 34th Ann. Int'l Symp. Microarchitecture* (Micro-34), IEEE CS Press, Los Alamitos, Calif., 2001 pp. 54-65.
12. D.C. Burger and T.M. Austin, *The Simplescalar Tool Set, Version 2.0*, tech. report CS-TR-1997-1342, Dept. of Computer Science, Univ. of Wisconsin, Madison, 1997.
13. M. Huang et al, "L1 Data Cache Decomosition for Energy Efficiency," *Proc. ACM/IEEE Int'l Symp. Low Power Electronics and*

Design, IEEE CS Press, Los Alamitos, Calif., 2001, pp. 10-13.
14. J.E. Smith, "A Study of Branch Prediction Strategies," *Proc. 8th Ann. Int'l Symp. Computer Architecture* (ISCA 81), IEEE CS Press, Los Alamitos, Calif., 1981, pp. 135-148.
15. T.Y. Yeh and Y.N. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction," *19th Ann. Int'l Symp. Computer Architecture* (ISCA 92), IEEE CS Press, Los Alamitos, Calif., 1992. pp. 124-134.
16. S.T. Pan, K. So, and J.T. Rahmeh, "Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation," *Proc. 5th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS-V), IEEE CS Press, Los Alamitos, Calif., 1992, pp. 76-84.
17. S. McFarling, *Combining Branch Predictors*, tech. report TN-36, Digital Equipment Corp., Western Research Lab, Palo Alto, Calif., 1993.
18. C. Zhang, X. Zhang, and Y. Yan, "Two Fast and High-Associativity Cache Schemes," *IEEE Micro*, vol. 17, no. 5, Sept.-Oct. 1997, pp. 40-49.
19. H. Kim et al., "Multiple Access Caches: Energy Implications," *Proc. IEEE Computer Soc. Ann. Workshop VLSI* (WVLSI), 2000, pp. 53-58.

Zhichun Zhu is a PhD candidate in computer science at the College of William and Mary. Her research interests include computer architecture, power-aware designs, and performance evaluation. Zhu has a BS in computer science from Huazhong University of Science and Technology, China. She is a student member of the IEEE and ACM.

Xiaodong Zhang is a professor of computer science at the College of William and Mary. He is also the program director of the Advanced Computational Research Program at the US National Science Foundation, Arlington, Virginia. His research interests include parallel and distributed systems, computer system performance evaluation, computer architecture, and scientific computing. Zhang has a BS in electrical engineering from Beijing Polytechnic University, China, and an MS and a PhD in computer science from the University of Colorado at Boulder. He serves on the editorial board of *IEEE Micro* and is a senior member of the IEEE.

Direct questions and comments about this article to Xiaodong Zhang, Dept. of Computer Science, College of William and Mary, Williamsburg, Va. 23187-8795; zhang@cs.wm.edu.