To appear in the **Proceedings of the 34th International Symposium on Microarchitecture (MICRO 34)**, 2001.

Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct-Mapping

Michael D. Powell[ϒ], Amit Agarwal[ϒ], T. N. Vijaykumar[ϒ], Babak Falsafi[†], and Kaushik Roy[ϒ]

[ϒ]*School of Electrical and Computer Engineering*
Purdue University
{mdpowell,amita,kaushik,vijay}@ecn.purdue.edu

[†]*Computer Architecture Laboratory*
Carnegie Mellon University
babak@cmu.edu

http://www.ece.purdue.edu/~icalp

Abstract

Set-associative caches achieve low miss rates for typical applications but result in significant energy dissipation. Set-associative caches minimize access time by probing all the data ways in parallel with the tag lookup, although the output of only the matching way is used. The energy spent accessing the other ways is wasted. Eliminating the wasted energy by performing the data lookup sequentially following the tag lookup substantially increases cache access time, and is unacceptable for high-performance L1 caches. In this paper, we apply two previously-proposed techniques, way-prediction and selective direct-mapping, to reducing L1 cache dynamic energy while maintaining high performance. The techniques predict the matching way and probe only the predicted way and not all the ways, achieving energy savings. While these techniques were originally proposed to improve set-associative cache access times, this is the first paper to apply them to reducing cache energy.

We evaluate the effectiveness of these techniques in reducing L1 d-cache, L1 i-cache, and overall processor energy. Using these techniques, our caches achieve the energy-delay of sequential access while maintaining the performance of parallel access. Relative to parallel access L1 i- and d-caches, the techniques achieve overall processor energy-delay reduction of 8%, while perfect way-prediction with no performance degradation achieves 10% reduction. The performance degradation of the techniques is less than 3%, compared to an aggressive, 1-cycle, 4-way, parallel access cache.

1 Introduction

High-performance caches dissipate significant dynamic energy due to charging and discharging of highly-capacitive bit lines and sense amps. As a result, caches account for a significant fraction of the overall chip dynamic energy. For instance, Pentium Pro consumes about 33% of chip power in instruction fetch and d-cache together [16], and Alpha 21264 consumes about 16% in caches [12].

To achieve low miss rates for typical applications, modern microprocessors employ set-associative caches. Fast, set-associative cache implementations probe the tag and data arrays in parallel, and then select the data from the matching way, which is determined by the tag array. At the time of precharging and reading the tag and data arrays, the matching way is not known. Consequently, conventional *parallel access* caches precharge and read *all* the ways but select *only one* of the ways on a cache hit, resulting in wasted dynamic energy dissipation. For example, a four-way set-associative cache discards three of the four ways on every access, wasting nearly 75% of the energy dissipated.

There are various options for reducing cache dynamic energy with different performance impact. The key to energy reduction is to pinpoint the matching way without probing all of the ways. One option to avoid high energy dissipation at the cost of slower access is by using *sequential access*, employed in the Alpha 21164's L2 cache [11]. In sequential access, the cache waits until the tag array determines the matching way, and *then* accesses only the matching way of the data array, dissipating about 75% less energy than a parallel access cache. Sequential access, however, serializes the tag and data arrays, adding as much as 60% to the cache access time [18]. The impact on the access time precludes sequential access for L1 caches.

In this paper, we apply two previously-proposed techniques, *way-prediction* [10,4] and *selective direct-mapping* [4], to reduce L1 dynamic cache energy while maintaining high performance. Way-prediction and selective direct-mapping predict the matching way number and provide the prediction *prior* to the cache access, instead of waiting on the tag array to provide the way number as done by sequential access. Predicting the matching way enables the techniques not only to attain fast access times but also to achieve energy reduction. The techniques reduce energy because *only* the predicted way is accessed. While these techniques were originally proposed to improve set-associative cache access times, this is the first paper to apply them to reducing energy.

The choices for d-cache way-prediction are to use information available either early in the pipeline, such as the program counter (PC), or later in the pipeline, such as an

XOR-based approximation of the load address [3]. Unfortunately, both choices have problems. Way-prediction based on information from early pipeline stages suffers from poor accuracy, and way-prediction based on late pipeline information introduces way-prediction table lookup delay in the cache access critical path [4]. For instance, the way-prediction scheme used in [13] inserts a table lookup after the address generation to identify the predicted way. Therefore, way-prediction is not nearly as effective for d-caches as it is for i-caches.

Fortunately, a majority (70% - 80%) of L1 d-cache accesses can avoid way-prediction altogether by using selective direct-mapping [4], and achieve low energy dissipation without performance loss. These accesses are the non-conflicting accesses (i.e., they do not map to the same set as another access) which can use direct mapping instead of set-associative mapping. In direct mapping, an address explicitly maps to only one of the N ways of a set-associative cache as if it were a direct-mapped cache; the address (and not the tag array) directly determines the matching way. Consequently, selective direct-mapping avoids the energy wastage of reading the other ways, without the need for way-prediction for the majority, i.e., non-conflicting, accesses. The rest of the accesses can be either way-predicted based on early available information with reasonable accuracy, or sequentially accessed with little performance loss.

I-cache way prediction can be integrated with branch prediction to achieve both high accuracy [9] and timeliness. Consequently, i-cache way-prediction achieves substantial energy reduction and obviates the need for selective direct-mapping for i-caches. The original proposal [9] way-predicts only taken branches, and we extend it to all instruction fetches, including not-taken branches, sequential fetches (i.e., non-branches), and function returns. These extensions allow us to achieve energy savings on nearly all i-cache accesses.

The main results of this paper are:

- Our techniques allow us to approach the performance of parallel access, set-associative caches with energy dissipation close to that of sequential access.
- Combining selective direct-mapping for d-caches and way-prediction for i-caches achieves an overall processor energy-delay reduction of 8%, compared to a 10% reduction assuming perfect way-prediction and no performance degradation. The performance degradation of the techniques is less than 3%, compared to an aggressive, 1-cycle, 4-way, parallel access cache.
- Selective direct-mapping supplemented with way-prediction achieves an average d-cache energy delay reduction of 69% over a 4-way parallel access L1 d-cache.

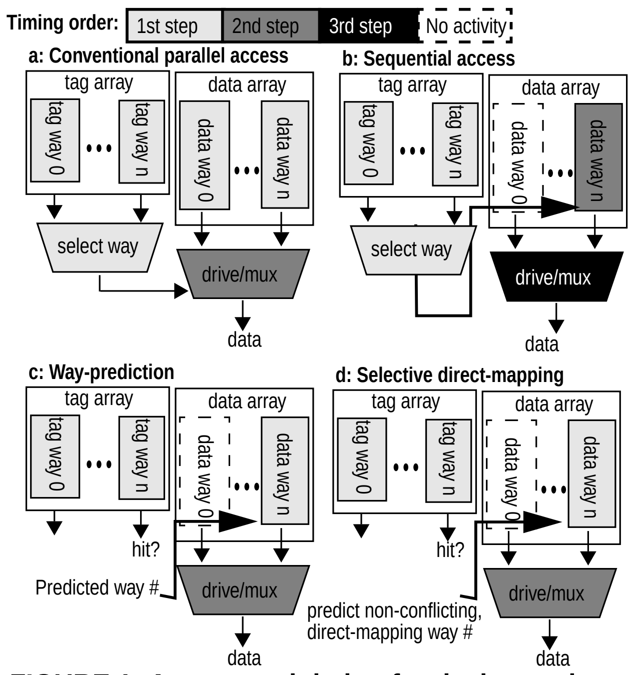

FIGURE 1: Access and timing for design options.

- Way-predicting i-cache accesses achieves an L1 i-cache energy delay reduction of 64% over a 4-way parallel access i-cache.

The rest of the paper is organized as follows. Section 2 describes how to reduce energy for cache accesses. Section 2.2 presents the selective direct-mapping framework for d-caches. Section 2.3 describes the way-prediction framework for i-caches. In Section 3 we describe the experimental methodology, and we present results in Section 4. Section 5 presents related work, and finally, we conclude in Section 6.

2 Reducing cache access energy

In this section, we discuss design considerations and the tradeoffs between energy and performance for the design options discussed in Section 1. Because the tag array is much smaller than the data array, energy disspiation in the data array is much larger than in the tag array. Therefore, we apply the energy optimizations considered in this paper only to the data array, and not the tag array.

2.1 Design options: performance and energy

Figure 1 shows the timing order of the actions involved in an access under each of the design options. Figure 1(a) depicts the relevant components and timing of a conventional, parallel access read. By reading the N data ways in parallel with the tag array, the set-associative design allows the data and select signal for the data-selection multiplexor to arrive at nearly the same time. The energy dissipated equals *tag array energy + N * (1 data way energy)*, and the access time equals *max (tag array time, data array time) + data mux time*.[1]

The above and the following energy and timing expressions are first-order approximations, but sufficient for our discussions. We include precharge in the energy expressions but do not include precharge in the timing expressions, because precharge time is not a part of access time [18], and no techniques we use impact precharge timing in any way.

The simplest solution to pre-determining which data way should be accessed is to wait for the output of the tag array, as done in sequential access and depicted in Figure 1(b). Sequential access deterministically yields the matching way number without resorting to prediction. Sacrificing performance by waiting to access only the correct way allows the design to achieve low energy on all accesses. The energy dissipated equals *tag array energy + (1 data way energy)*, and the access time equals *tag array time + data array time + data mux time*. Compared to a parallel access cache, the sequential access dissipates less energy of the amount *(N-1) * (1 data way energy)*, but is slower by *min (tag array time, data array time)*. Typically, the access speed difference between sequential and parallel access is about 60%.

An alternative design that only accesses the one way on most access is to employ way-prediction as depicted in Figure 1(c). We describe details of how the prediction is made in Section 2.2.1, but focus on the access timing and energy aspects here. Upon an access, the tag array and the predicted data way are probed simultaneously. On a hit, the tag array determines that either the predicted data way holds the data and flags a correct way-prediction or another data way holds the data and flags a misprediction. On a misprediction, the data array is accessed again, probing the correct data way as determined by the tag array. Thus, correctly way-predicted accesses are as fast as a parallel access with the low energy of a sequential access. But mispredicted accesses increase both energy and access time due to the second probe. If the prediction accuracy is high, the energy and timing penalty of the second probe is tolerable.

As outlined in Section 1, selective direct-mapping (*selective-DM*) solves way-prediction's inability to achieve both high accuracy and timely prediction. Selective-DM isolates non-conflicting accesses and explicitly maps them to only one of the N ways, as if it were a direct-mapped cache. The address (and not the tag array) directly determines the matching way obviating the need for way-prediction. The isolation is achieved by predicting that an access is non-conflicting and we describe the details in Section 2.2.2. While way-prediction probes the predicted way, selective-DM predicts that an access is non-conflicting and probes its direct-mapping way. The direct-mapping way is identified by the address's index bits extended with $\log_2 N$ bits borrowed from the tag.

Selective-DM access, depicted in Figure 1(d), proceeds similar to a way-predicted access. A correctly predicted access probes only the matching data way, and a misprediction initiates a second probe of the correct data way. Because most accesses are non-conflicting and are captured by selective-DM, a high energy or low performance solution, such as parallel or serial access respectively, may be acceptable for conflicting accesses. For designs where conflicting accesses also need to be low energy and high performance, way-prediction can be employed. In our experiments, we evaluate several such combinations.

It is apparent from Figure 1(c) and Figure 1(d) that the energy and timing are similar for way-prediction and selective-DM. For a correctly predicted access in either scheme, the energy dissipated equals *tag array energy + (1 data way energy)*, and the access time equals *max (tag array time, data array time) + data mux time*. For a mispredicted access, the second probe increases the energy by *(1 data way energy)* and extends the access time by *data array time.* With either scheme, mispredictions incur a latency penalty because they require two data array probes. But because only two data ways are accessed (the mispredicted way and the correct way) in all, the total energy of a misprediction is not as high as that of a parallel access when set-associativity is greater than two.

Both way-prediction and selective direct mapping require predicting the behavior of accesses. Because stores change program state, store accesses cannot use predicted information. Therefore, way-prediction and selective-DM apply only to loads. Stores check the tag array first to determine the matching way and then probe and write into only the matching way, even in conventional parallel access caches. Thus, stores do not waste energy and do not need way-prediction or selective direct mapping.

2.2 Way-prediction and selective-DM for d-caches

In this section, we discuss potential prediction sources and needed structures for using the schemes on d-caches.

2.2.1 Way-prediction

In general, way-prediction schemes look up a prediction table using a handle to index into the table and obtain the

1. Each data and tag way may further be divided into subarrays in a modern cache design. Subarrays do not impact the energy wastage associated with reading the N ways in a set-associative cache; therefore we do not show subarrays in the figure, nor include subarrays in our discussions.

predicted way number. The predicted way number must be made available before the actual data address to avoid any delay in the initiation of every cache access. This stipulation rules out the technique in [13] that uses the data address as the handle. Two viable handles are: the load PC and approximate data address formed by XORing the load's source register with the load offset. The XOR-approximation was proposed in [3] for zero-cycle loads, and used in [10] to improve set-associative d-cache access times.

These two handles represent the two extremes of the trade-off between prediction accuracy and early availability in the pipeline. The PC is available much earlier than the XOR approximation but the XOR approximation is more accurate. XOR operation on a register value often obtained late from a register-forwarding path followed by a table lookup, is likely to be slower than a full add to compute the address, delaying access initiation. In contrast, PC-based way-prediction allows at least six pipeline stages (fetch through execute stages) for the lookup, making the predicted way-number available well before the data address. The PC-based scheme relies on a load accessing the same block repeatedly. Previous studies have shown that such per-instruction block locality is fairly prevalent due to common code patterns [4]. For instance, a load in a loop accessing different words of a cache block (e.g., sequential array elements) in different iterations, or a load in a loop accessing the same word in a block in different iterations (e.g., a global static variable).

2.2.2 Optimizing selective-DM: identifying conflicts

Selective-DM requires isolating conflicting from non-conflicting cache blocks so that non-conflicting accesses are placed in direct-mapping ways. Based on this isolation, accesses need to be flagged to use direct-mapping for non-conflicting blocks and set-associative mapping for conflicting blocks. This framework is depicted in Figure 2. Set-associative mapping may further employ parallel, sequential, or way-prediction to achieve an acceptable performance-energy point.

To identify conflicting blocks, we count the number of times a block is evicted. Cache blocks are considered non-conflicting by default and are placed in their direct-mapping way. We identify conflicting blocks by maintaining a list of victim (i.e., replaced) block addresses. On a replacement, the evicted block increments its entry's counter in the victim list if it is already present in the victim list; otherwise, a new victim list entry is allocated. If the count exceeds two, the block is deemed conflicting and placed in its set-associative position to avoid future conflicts. Previous papers have shown that using a victim list captures replacements occurring within a short duration, and effectively identifies conflicts.

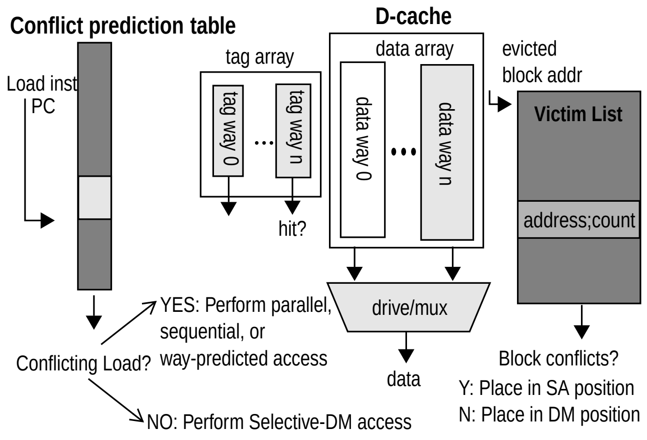

FIGURE 2: Prediction framework for selective-DM.

Flagging an access to use direct mapping or set-associative mapping is similar to way-prediction in its timing requirements. The identification of the mapping needs to be made early in the pipeline to avoid any delay in the cache access, as discussed in Section 2.2.1. Because this identification also uses a table lookup much like way-prediction, we use the PC to look up the mapping to be used for the access. Previous studies have shown that most misses are caused by a few instructions [1], implying that identifying the few will allow us to use direct mapping for the rest of the instructions. Unlike way-prediction, which requires for each load instruction a prediction of one of N options for an N-way set-associative cache, selective direct mapping requires only a binary choice of using direct-mapping or set-associative mapping. This binary choice allows us to achieve accurate prediction using the PC.

We maintain a prediction table indexed by the load PC with each entry consisting of a two-bit counter with values saturating at 0 and 3. The counter value is used to determine if the cache should be probed using direct or set-associative mapping. If a cache read results in a hit using the direct-mapping way, the counter is decremented. If a read results in a hit using a set-associative mapping way, the counter is incremented. Counter values of 0 and 1 flag direct-mapping, and values 2 and 3 flag set-associative mapping.

Both selective-DM and way-prediction use simple lookup tables to perform their prediction. Because the tables are small with respect to the caches, their energy overhead is small; however, we account for the overhead in our results in Section 4.

2.3 Way-prediction for i-caches

Timely and accurate way-prediction for i-caches can be implemented by extending branch prediction concepts. The fetch hardware performs branch prediction to determine the next PC while accessing the i-cache with the current

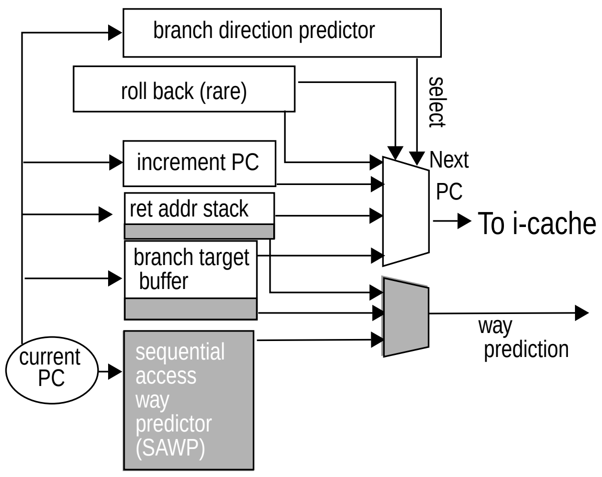

FIGURE 3: Fetch and i-cache access mechanism.

PC. Because i-cache accesses occur at the beginning of the pipeline, we use the PC of the previous access for way-prediction. Way-prediction is performed along with branch prediction so that way-prediction does not add any delay to the i-cache access. By the time the previous i-cache access is complete, the next predicted PC and the predicted way are ready.

Existing high-performance processors use a branch target buffer (BTB) to determine the next fetch address for predicted taken branches. Next-line-set-prediction supplies a way-prediction for taken branches [9]. We extend this concept to provide way-predictions when the next address does not come from the BTB. For not-taken branches and sequential fetches (non-branches), we use an extra table called the Sequential Address Way-Predictor (SAWP) table, which is indexed by the current PC. At first glance, the SAWP might seem unnecessary, because the incremented PC would map to the same way as the current PC. However, successive PCs may not fall within the same way. For function returns, we augment the return address stack (RAS) to provide not only the return address but also the return address's way. Figure 3 depicts the fetch hardware. Shaded components are not present in a conventional system but are part of our way-prediction mechanism.

On a branch misprediction, the correct next PC comes from branch resolution in the processor. There is not enough time between the resolution and the next fetch to look up a way-prediction structure. Because branch mispredictions are infrequent, our scheme defaults to parallel access. It is also possible that the way-prediction structures do not return a prediction (i.e., the current PC misses in the way-prediction structures). In that case too, our scheme defaults to parallel access.

A correctly way-predicted fetch (irrespective of whether it is correctly branch predicted or not), accesses the tag array and the predicted data way. Way mispredicted fetches probe the matching data way a second time, incurring extra energy and access time, exactly as in d-caches. The energy and access time are identical to those of d-cache way-prediction given in Section 2.1. In our results, we show that, unlike d-caches, i-caches exhibit way-prediction accuracy high enough to make energy and time penalty of way mispredictions small. The small penalty means we do not need selective-DM for i-caches. Our scheme adds log n bits to each entry of the BTB, SAWP and RAS for an n-way set-associative i-cache. In our results, we account for the energy overhead due to this addition.

Table 1: System configuration parameters.

Instruction issue & decode bandwidth	8 issues per cycle
L1 i-cache	16K, 4-way, 1 cycle
Base L1 d-cache	16K, 4-way, 1 or 2 cycles, 2 ports
L2 cache	1M, 8-way, 12 cycle latency
Memory access latency	80 cycles + 4 cycles per 8 bytes
Reorder buffer size	64
LSQ size	32
Branch predictor	2-level hybrid

3 Methodology

Table 1 shows the base configuration for the simulated systems. The papers that explored way-prediction [10] and selective-DM [4] show that the access time of a set-associative cache employing these techniques is less than that of a parallel access cache. Because we do not want to revisit this aspect and we want to isolate energy reduction from access time improvements in calculating energy-delay, we conservatively assume that these techniques do not change access times. Note that this assumption does not give our energy-saving techniques any advantage, but actually accentuates any performance loss incurred by them. In Section 4.1, Section 4.2, and Section 4.3, we compare our techniques against an aggressive, 1-cycle, 4-way set-associative parallel access cache, and in Section 4.4, we compare against a more realistic 2-cycle cache.

We modify Wattch [6] and incorporate SimpleScalar 3.0b [8] modifications to simulate a high-performance out-of-order microprocessor executing the Alpha ISA. We estimate overall processor energy using Wattch and cache and prediction table energy using Cacti scaled for a 0.25 micron process [18]. To ensure that cache circuitry is compatible with parallel, serial, direct-mapped, and sequential accesses, all caches use set-associative geometry for the address decoding and output logic. In configurations using way-prediction or selective direct mapping, our prediction tables have 1024 entries. In systems using selective direct mapping, our victim list has 16 entries. Table 2 shows the benchmarks used for our study, the corresponding inputs,

Table 2: Applications and input sets.

name	input	#of inst(billions)
Integer benchmarks		
gcc	ref	0.345
go	ref	1.07
li	train	0.207
m88ksim	train	0.135
perl	train	1.07
troff	train	0.051
vortex	test	1.07
Floating point benchmarks		
applu	train	1.07
fpppp	train	0.234
mgrid	train	1.07
swim	test	0.492

and the number of dynamic instructions executed.

Modern caches typically use subarrays at the circuit-level to optimize for speed and energy. We assume an energy-efficient baseline cache, which activates *only* the subarrays containing the addressed set (including all the ways in the set) and not all the subarrays. Subarrays, however, cannot reduce the energy wasted in reading all the ways of a conventional set-associative cache. Therefore subarrays, even if energy efficient, do not nullify the savings achieved by our techniques. It is possible that activating only the subarrays containing the addressed set may encroach upon timing constraints. In that case, additional opportunities to save the resulting subarray energy wastage exist, and may be exploited by extending our techniques. Such extensions are beyond the scope of this paper.

We also modify Cacti to compute the energy component of our prediction structures. This component is included in all the relevant energy calculations but is always less than 1% of the conventional d-cache energy because our prediction tables and victim lists are small (less than 1 K and 0.06 KB respectively). We include the tag array energy, which is about 6% of the conventional d-cache energy, in *all* calculations even though our techniques optimize only the data array energy and not the tag array energy, as mentioned at the beginning of Section 2. Similarly, we include both loads and stores in all calculations, although our optimizations do not apply to stores.

4 Results

We present cache energy-delay in Section 4.1 through

Table 3: Cache energy and prediction overhead.

Energy component	Relative Energy
Parallel access cache read (4 ways read)	1.00
Sequential-access, way-predicted, or direct-mapping access (1 way read)	0.21
Cache write	0.24
Tag array energy (also included in all above rows)	0.06
1024 entry x 4 bit prediction table read/write	0.007

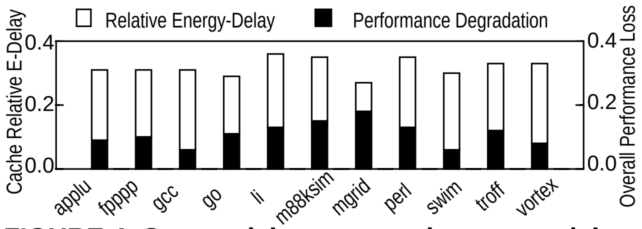

FIGURE 4: Sequential-access cache energy-delay.

Section 4.4, and overall processor energy-delay in Section 4.5. Section 4.1 shows that sequential access saves energy at the cost of performance and hence is an inadequate solution for energy savings. Section 4.2 shows that way-prediction based on early available PC is inaccurate and does not reduce energy-delay much, and way-prediction based on late available XOR approximation may impact timing. Section 4.3 shows that selective-DM based on PC significantly reduces energy without introducing timing problems. In Section 4.4, we vary cache size, set-associativity, and latency, showing that selective-DM is an effective means of energy reduction for larger and slower caches and that the energy savings increase with set-associativity. In Section 4.5, we show that way-prediction is effective for i-caches. Finally, in Section 4.5, we show that combining selective-DM for d-caches and way-prediction for i-caches reduces overall processor energy-delay by 8%.

4.1 Energy savings in sequential access d-caches

In this section we discuss performance and energy considerations for a sequential access, assuming two cycles per access. Because this configuration avoids the energy wastage of reading multiple ways during a cache access, we expect significant energy savings over a parallel access cache. However, we expect sequential access caches to incur significant performance degradation due to their respectively higher miss ratios and longer access latency.

Cacti simulations indicate that avoiding parallel reads results in significant energy savings. Table 3 shows cache energy dissipation for parallel and sequential access cache reads as well as cache writes. As expected, the parallel read dissipates approximately four times as much energy as the sequential read. It is worth noting that the write energy is not affected by the read configuration because all cache writes probe only one way, as mentioned at the end of Section 2.1.

While sequential access allows all cache reads to access only one way, the energy savings comes at the expense of performance. Figure 4 shows energy-delay and performance for a sequential-access cache relative to a 1-cycle, parallel access cache. The black bars represent performance degradation; the lighter bars represent relative d-cache energy delay (i.e., relative d-cache energy multiplied by relative execution time for each application). While

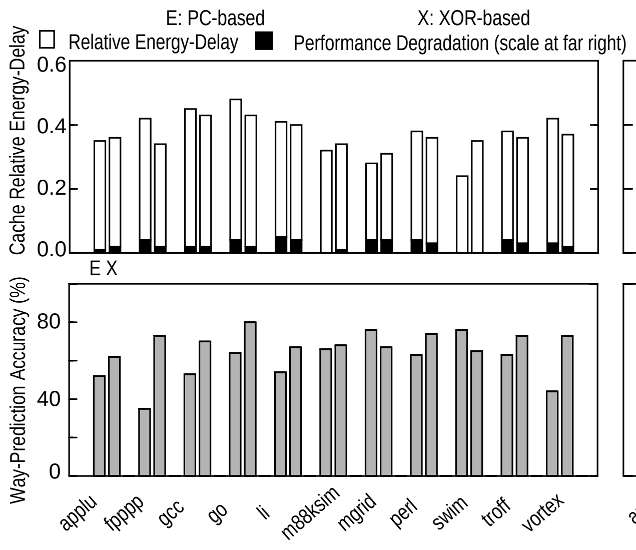

FIGURE 5: PC- and XOR-based way-prediction.

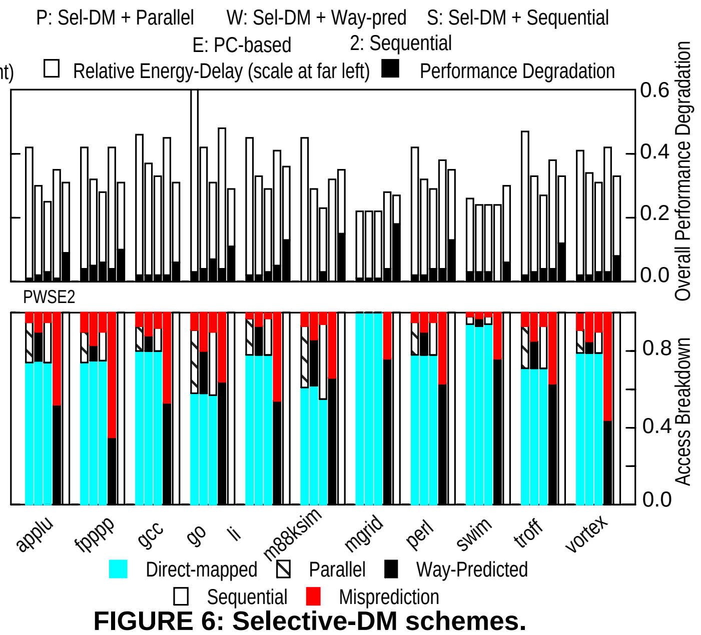

FIGURE 6: Selective-DM schemes.

sequential access achieves energy-delay savings of 68% over a parallel access cache by always accessing only one way, the average performance degradation 11% (maximum degradation is 18%) for the sequential access cache.

The primary reason for this degradation is the substantial latency increase for the sequential access cache. that occurs because *every* access takes two cycles. This weakness means sequential access is not an acceptable solution for energy reduction in high performance d-caches.

4.2 Energy savings in way-predicted d-caches

In this section we discuss the accuracy, energy, and timing considerations for two forms of way-prediction. Way-prediction may be based on either early available or late available information, as discussed in Section 1. While use the early available PC ensures timely way-prediction, we expect PC-based way-prediction to experience low accuracy. Late available XOR-based approximation of the block address should have higher accuracy. We assume an additional cycle for mispredicted accesses.

Figure 5 shows our results for PC-based and XOR-based way-prediction. The light and dark bars of the top graph represent relative energy-delay and relative performance with respect to a 1-cycle, parallel access cache. The bottom graph depicts prediction accuracy for each scheme.

PC-based way-prediction has an average accuracy of 60% in contrast to XOR-based prediction's 70%. The difference is consistent with the fact that the PC does not provide information about the actual address and is most effective in exploiting per-instruction block locality. In contrast, the XOR of the load address components is a reasonable approximation of the block address. The benchmarks with the lowest accuracy in the XOR-based scheme are *applu*, *mgrid,* and *swim,* which have the highest cache miss rates of 7%, 5%, and 25% respectively.

The differences in relative energy-delay between PC based and XOR based way-prediction are primarily due to variation in prediction accuracy and the associated performance degradation. The average performance degradations are 2.9% and 2.3% for each scheme, while the average relative energy-delay reductions are 63% and 64%. The fairly low performance degradations occur because most of the applications can overlap the additional latency of a small number of mispredicted d-cache accesses. However, the lower energy-delay reduction of PC-based way-prediction, compared to the 68% energy-delay reduction of sequential access cache, makes it a suboptimal solution for energy reduction. A larger prediction table does not improve the PC-based scheme's accuracy (energy-delay and performance change less than 1% for a 2048 entry table), and hence does not improve energy-delay reduction.

While XOR-based way-prediction is more accurate, the technique must meet a difficult timing constraint in order to avoid impacting the cache critical path. For a 1024-entry prediction table, the size suggested in [10], Cacti estimates that table lookup time is 48% of the 16K 4-way cache access time itself. The lookup time is likely to be larger than the time required to compute the actual address via a full add, delaying the cache access. This result makes the XOR scheme hard to implement in high-performance systems.

4.3 Obviating way-predictions with selective-DM

In this section, we evaluate the use of selective-DM to

Table 4: D-cache miss rates.

Technique	*applu*	*fpppp*	*gcc*	*go*	*li*	*m88ksim*	*mgrid*	*perl*	*swim*	*troff*	*vortex*
Direct-mapped	8.2	6.3	5.1	5.9	4.7	3.5	5.4	3.0	23.3	2.7	3.1
4-way Set-associative	7.0	0.5	3.3	2.0	3.3	1.3	5.1	1.3	25.2	0.8	1.8

reduce d-cache energy and obviate the need for way-prediction for most accesses. Selective-DM uses the PC to predict direct-mapping placement for the majority non-conflicting d-cache accesses, avoiding the timing and accuracy concerns of way-prediction. The non-conflicting accesses probe only the direct-mapping way, avoiding the energy wastage of parallel access. For the remaining conflicting accesses, we evaluate the use of parallel, way-predicted, and sequential access for their impact on energy and performance. We expect the combination of selective-DM and way-prediction to approach the low energy dissipation of a sequential access cache while maintaining performace close to that of a parallel access cache. As before, a mispredicted access due to either way-prediction or selective-DM takes an extra cycle.

Figure 6 depicts our results for selective-DM. As in earlier sections, the top graph shows energy-delay and performance degradation relative to a system using a 1-cycle, parallel access cache. The leftmost bar for each application represents selective-DM in combination with parallel access. The second bar depicts selective-DM supplemented with PC-based way-prediction for the remaining accesses. The third bar shows selective-DM with sequential access for conflicting cache reads. The fourth and fifth bars are PC-based way-prediction and sequential access; they are repeated from earlier graphs for reference. The bottom graph breaks down the various types of cache accesses: direct-mapped, parallel, way-predicted, sequential, and mispredicted, which includes both incorrect way-predictions and accesses mispredicted as direct-mapped.

Selective-DM correctly predicts an average of 77% of all d-cache reads as non-conflicting accesses. Such accuracy is reasonable considering: (1) selective-DM uses PC to predict, and (2) selective-DM identifies more than 60% of cache accesses as non-conflicting even for applications requiring set-associativity. This claim is borne out by Table 4, which shows that there are significant differences in the miss rates between direct-mapped and conventional 4-way set-associative cache except for *swim* which experiences pathological misses in the 4-way cache. As was the case for PC-based way-prediction, increasing the size of the prediction table does not impact the prediction accuracy of selective-DM.

Unfortunately, using parallel access for the conflicting accesses incurs high energy dissipation. These parallel accesses are the primary contributor to the low energy-delay reduction (average of 59%) and low performance degradation (average of 2.0%) of this configuration. Only conflicting accesses mispredicted as DM, averaging 6% of cache reads per application, incur a latency penalty. This lower penalty is a significant improvement over PC-based way-prediction which consumes an extra cycle on an average of 40% of accesses due to low accuracy. All DM accesses correctly predicted as non-conflicting and all parallel accesses predicted conflicting have a latency of one cycle.

The high energy parallel access energy of the conflicting accesses may be eliminated in one of two ways. The conflicting accesses may be converted to sequential access or way-predicted. The simplest choice is to use selective-DM in combination with sequential access, achieving an average energy-delay reduction of 73% with an average performance degradation of 3.4%. An incremental extension to this scheme adds a way number to the prediction table, allowing way-prediction instead of sequential access. Adding PC-based way-prediction decreases average performance degradation to 2.4% but also decreases average energy-delay reduction to 69%. Both of these configurations achieve energy-delay savings above the 68% achieved by sequential access, without incurring its 11% performance degradation.

Of the two configurations discussed above, selective-DM plus way-prediction is slightly better for performance because it maintains one-cycle latency for correctly way-predicted accesses. However, each additional misprediction causes an energy-wasting second probe. Conversely, selective-DM plus sequential access is the better choice for energy because all conflicting accesses dissipate low energy in this scheme. The only two exceptions are *mgrid* and *swim*, in which the configurations with sequential access and way-prediction have energy delays within 1% of each other. For *mgrid,* over 99% of cache accesses are nonconflicting; therefore the choice of handlers for conflicting accesses is irrelevant. For *swim,* the d-cache miss rate is approximately 25% so the L2 cache latency controls performance more than the extra cycle required to retrieve conflicting cache hits.

Table 5: D-cache summary.

Technique	% Avg. E-delay savings	% Avg. Perf. Loss	Problem
Sequential-Access cache	68	11	high perf. degradation
PC-based way-prediction	63	2.9	low e-savings
XOR-based way-prediction	64	2.3	timing
Sel-DM +parallel access	59	2.0	low e-savings
Sel-DM +way-prediction	69	2.4	
Sel-DM +sequential access	73	3.4	

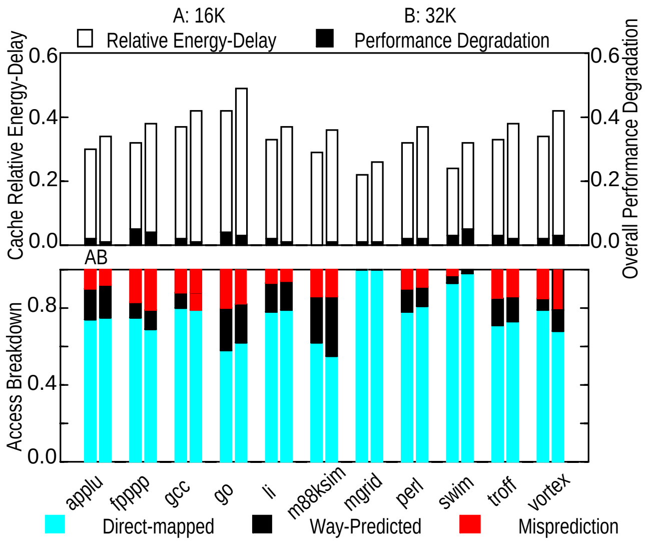

FIGURE 7: Effect of cache size on selective-DM.

In Table 5, we summarize the energy-delay and performance of the various d-cache design options. From this table it is clear that selective-DM supplemented with sequential access or way-prediction achieve the highest energy-delay reduction with least performance degradation. The rest of the options achieve low energy-delay reduction or high performance degradation, or have critical path timing problems.

4.4 Effect of cache parameters on selective-DM

In this section we evaluate the effect of varying conventional cache parameters on selective-DM schemes. First, we discuss the impact of increasing the cache size to 32K. Second, we evaluate the energy savings for 2-way and 8-way set-associative caches in addition to the 4-way caches already studied. Finally, we evaluate the impact of increasing the base d-cache latency from 1 cycle to 2 cycles.

We do not expect an increase in cache size to impact substantially the opportunity for energy savings. Figure 7 compares the results for selective-DM plus PC-based way-prediction for 16K and 32K caches. Relative energy-delay is computed with respect to a 1-cycle, parallel access 4-way cache of the same size. The results indicate that the 32K cache achieves an average energy-delay savings of 63% with an average performance degradation of 2.1%. These numbers are similar to the averages of 69% and 2.4% for 16K caches. The primary reason for the decrease in savings as size increases is because certain energy components not affected by our scheme, such as tag energy and address decode, increase slightly as proportion of total cache energy. It is also worth noting that the misprediction rate did not increase even though we did not scale the size of the 1024 entry prediction table for the 32K cache. The reason is our prediction is based on the PC, not the block address, and the table does not need additional entries for the added cache sets. Additional simulations (not shown) indicate that increasing the size of the prediction table to 2048 entries has minimal effect on performance or energy (< 0.2% change for both) for the 32K cache.

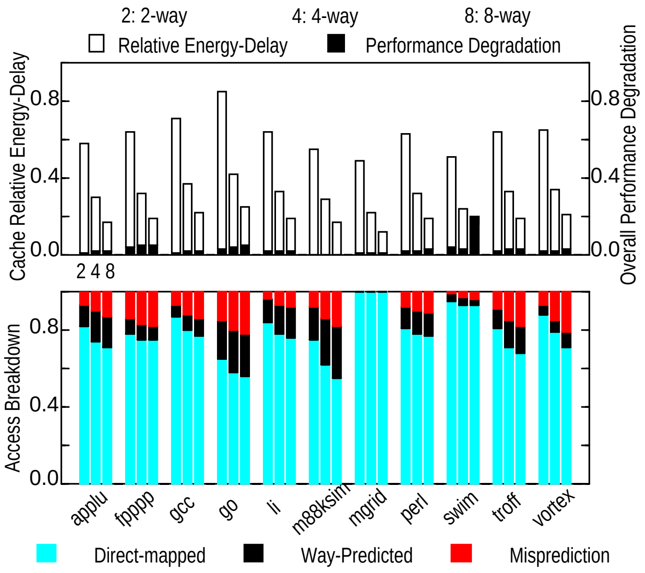

FIGURE 8: Effect of associativity on selective-DM.

Unlike size, increasing set-associativity significantly increases the opportunity for energy savings. As discussed in Section 2, an N-way parallel access cache wastes energy by accessing (N-1) more ways than necessary. Figure 8 shows results for 16K 2-way, 4-way, and 8-way set-associative caches. Relative energy-delay is computed with respect to a 1-cycle, parallel access cache of the same associativity. The graph clearly depicts the trend of increasing energy savings; the average energy-delay reduction for 2-way, 4-way, and 8-way caches is 38%, 69%, and 82% respectively. One problematic application, *swim*, experiences substantial performance degradation in the 8-way case because pathological conflicts cause the miss rate to increase more than 5% above that of a parallel access 8-way cache.

Two additional trends relating to increased set-associativity are apparent from the bottom graph of Figure 8. The first is that the number of mispredictions increases slightly with set-associativity because a random way-prediction is less likely to be correct when there are more ways. The second trend is that the number of non-conflicting accesses does not decrease dramatically with increasing set-associativity, helping selective-DM to scale well. The fact that most accesses remain non-conflicting allows selective-DM to exploit the increased energy-saving opportunity in highly associative caches.

Finally, we show results for selective-DM when the base d-cache latency is increased to 2 cycles. As interconnect latencies become more significant, microprocessors

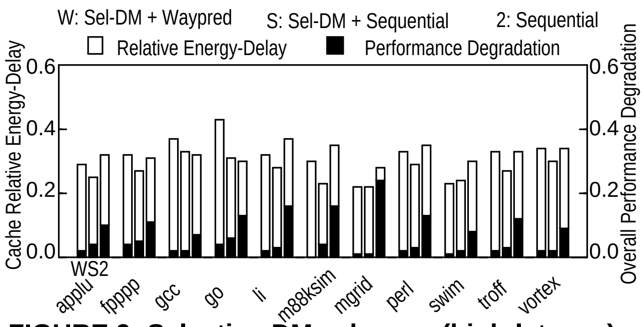

FIGURE 9: Selective-DM schemes (high-latency).

have trended toward 2 cycle d-cache accesses where one cycle consists of wire delay. When selective-DM plus way-prediction, Selective-DM plus sequential access, or sequential access is applied in this situation, a mispredicted or sequential access requires a total of 3 cycles. An out-of-order engine may not be able to absorb this latency without impacting performance.

Figure 9 depicts results for selective-DM plus way-prediction, selective-DM plus sequential access, and sequential access for a 2 cycle d-cache. Energy-delay values are relative to a 2-cycle, parallel access cache. Selective-DM plus way-prediction and Selective-DM plus sequential access achieve average energy-delay savings of 69% and 73% respectively with average performance degradation of 2.0% and 3.1% respectively. These results are similar to those for the 1 cycle cache, and the small performance degradation indicates the system can absorb the additional latency of some three cycle accesses. In contrast, sequential access incurs performance degradation of nearly 13%, 2% worse than with a 1 cycle cache. This additional degradation indicates that the system cannot absorb the latency of *all* d-cache access being 3 cycles without substantial performance impact.

4.5 Way-prediction for i-caches

In this section we evaluate energy reduction for i-caches using way-prediction as discussed in Section 2.3. Because i-cache way-prediction extends from existing, highly accurate fetch address prediction, we expect to capture nearly all accesses and achieve high energy savings. As with d-caches, we expect energy savings to increase with set-associativity.

Figure 10 shows results for 16K 2-way, 4-way, and 8-way i-caches using a 1024 entry SAWP. As in previous sections, the top graph depicts energy-delay and performance degradation relative to a 1-cycle, parallel access i-cache. From bottom to top, the subbars on the bottom graph represent accesses correctly predicted by the SAWP, accesses correctly predicted using the branch predictor (BTB and RAS), parallel accesses not predicted, and mispredicted accesses.

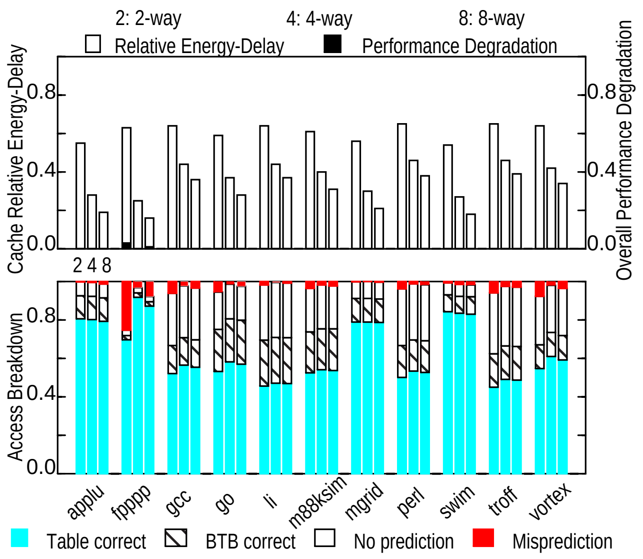

FIGURE 10: Way-prediction for i-caches.

From the bottom graph, we see that overall prediction accuracy is high. Accuracy is greater than 92% for all applications except *fpppp*, which has a large, conflicting code footprint that thrashes in the i-cache, lowering way-prediction accuracy. As indicated by the graph, the branch predictor most strongly contributes to correct predictions in branch-intensive integer applications such as *go, li,* and *m88ksim*. Floating point applications with longer basic blocks, such as *applu, mgrid,* and *swim* use SAWP for well over 75% of i-cache accesses. The only accesses not predicted by our scheme are BTB misses and misprediction restarts. Predicting such a wide range of accesses gives our scheme an energy advantage over alternatives such as Next-line-set-prediction, which only way-predicts taken branches.

The top graph indicates we achieve high energy-delay savings with negligible performance degradation. For 2-way, 4-way, and 8-way set-associative caches, average energy-delay savings are 39%, 64%, and 72% respectively. As with d-caches, energy-delay savings increases significantly with set-associativity because of increased opportunity for energy savings. Performance degradation is less than 0.5% for all applications except *fpppp*, which incurs high misprediction rates. Because of high prediction accuracy and minimal performance degradation, way-prediction is highly effective in reducing i-cache energy.

4.6 D- and I-cache: processor energy-delay savings

In this section, we present results for overall processor energy-delay when selective-DM plus way-prediction for d-caches is combined with i-cache way-prediction. Our simulations indicate L1 i-caches and d-caches dissipate 10% to 16% of the overall processor energy, depending on the application. Out of this achievable overall energy sav-

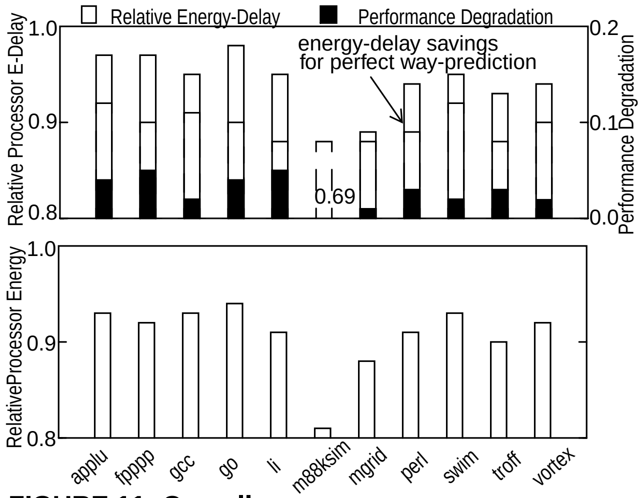

FIGURE 11: Overall processor energy.

ings, performance degradation lessens actual savings due to the energy dissipated by the processor in the extra cycles.

Figure 11 depicts relative energy-delay, performance degradation, and relative overall energy. The top graph shows energy-delay and performance degradation relative to a processor using 1-cycle, parallel access i- and d-caches; the bottom graph shows relative energy. For d-caches, we use selective-DM plus way-prediction as in Section 4.3. For i-caches, we use way-prediction as in Section 4.5. Note that there are two differences with respect to previous graphs: relative energy-delay and performance degradation are on different scales, and relative energy-delay starts from 0.8.

Our average relative energy savings is 9% with most applications experiencing savings between 6% and 12%. Taking into account performance degradation, the average relative energy-delay reduction is 8%, compared to a 10% reduction assuming perfect way-prediction and no performance degradation. Most applications experience energy-delay savings between 3% and 10%.

The only exception among the applications is *m88ksim*, which experiences a pathological speedup as a result of i-cache way-prediction. Way-prediction misses in the BTB allow additional time for branch resolution, substantially increasing branch prediction accuracy and resulting in a 15% speedup. None of our other applications exhibit this behavior.

5 Related Work

Selective cache ways [2] pioneered reducing cache energy by turning off unneeded ways in a set-associative cache. Our techniques are orthogonal to selective cache ways in that our techniques can be used on the ways that are left active by selective cache ways. Apart from this orthogonality, there are important differences. First, selective cache ways is a coarse-grain approach which decides whether an entire application requires set associativity or not. In contrast, selective-DM decides whether each access requires set-associativity or not (i.e., conflicting or not). If an application uses all the cache ways, selective cache ways cannot achieve any energy savings but selective-DM can filter out the majority non-conflicting accesses. For instance, using a 32K 2-way d-cache, selective cache ways cannot achieve any savings in six of the eight benchmarks, including *fpppp*, with less than 4% performance loss [2]. Selective-DM plus way-prediction achieves 38% energy-delay reduction for a 16K 2-way d-cache, including 37% for *fpppp*, with similar performance loss. Note that it is harder to achieve energy savings in a smaller cache without impacting performance, and selective-DM achieves more relative energy savings for a smaller cache. Second, selective cache ways requires manual or software configuration to choose the number of active cache ways per program, but selective-DM is a transparent, hardware-only technique.

Other proposals that specifically focused on reducing cache energy dissipation. Many of these propose placing small energy-efficient buffers (e.g., 128-256 bytes in size) in front of caches to filter traffic to the cache. Examples include block buffers [7,17], filter cache [15], and loop cache [5]. Filtering, however, is only effective when application working sets are extremely small and is otherwise not applicable. Moreover, when working sets do not fit, filtering increases the critical path access time for accessing cached information and may significantly reduce performance. To reduce leakage energy dissipation, Powell, et al. [19], propose a dynamically resizing i-cache, and Kaxiras et al. [14] propose using cache decay. These papers do not address dynamic energy, but only leakage energy.

6 Conclusions

Set-associative caches minimize access time by accessing all the data ways in *parallel* with the tag lookup, although the output of only the matching way is used. The energy spent accessing the other ways is wasted. The key to energy reduction is to pinpoint the matching way without probing all of the ways. Eliminating the wasted energy by performing the data lookup *sequentially* following the tag lookup substantially increases cache access time, and is unacceptable for high-performance L1 caches.

In this paper, we applied two previously-proposed techniques, *way-prediction* and *selective direct-mapping,* to reducing L1 cache dynamic energy while maintaining high performance. Way-prediction and selective direct-mapping predict the matching way number and provide the prediction *prior* to the cache access, instead of waiting on the tag array to provide the way number as done by sequential access. Predicting the matching way enables the techniques not only to attain fast access times but also to achieve energy reduction. The techniques reduce energy because *only* the predicted way is accessed. While these techniques were originally proposed to improve set-associative cache access times, this is the first paper to apply them to reducing energy.

We evaluated the effectiveness of these techniques in reducing L1 d-cache, L1 i-cache, and overall processor energy. Using these techniques, our caches achieve the energy-delay of sequential access while maintaining the performance of parallel access. Relative to parallel access L1 i- and d-caches, the techniques achieve overall processor energy-delay reduction of 8%, while perfect way-prediction with no performance degradation achieves 10% reduction. This overall reduction combines L1 d-cache energy-delay reduction of 69% and L1 i-cache energy-delay reduction of 64%. The performance degradation of the techniques is less than 3%, compared to an aggressive, 1-cycle, 4-way, parallel access cache.

Acknowledgements

This research is supported in part by SRC under contract 2000-HJ-768, DARPA under contract DAAH04-96-1-0222 and an NSF Graduate Research Fellowship.

References

[1] S. G. Abraham, R. A. Sugumar, D. Windheiser, B. R. Rau, and R. Gupta. Predictability of load/store instruction latencies. In *Proceedings of the 26th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 26)*, pages 139–152, Dec. 1993.

[2] D. H. Albonesi. Selective cache ways: On-demand cache resource allocation. In *Proceedings of the 32nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 32)*, pages 248–259, Nov. 1999.

[3] T. M. Austin and G. Sohi. Zero-cycle loads: Microarchitecture support for reducing load latency. In *Proceedings of the 28th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 28)*, Dec. 1995.

[4] B. Batson and T. N. Vijaykumar. Reactive associative caches. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compiliation*, Sept. 2001.

[5] N. Bellas, I. Hajj, and C. Polychronopoulos. Using dynamic management techniques to reduce energy in high-performance processors. In *Proceedings of the 1999 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 64–69, Aug. 1999.

[6] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.

[7] J. Bunda, W. Athas, and D. Fussell. Evaluating power implications of CMOS microprocessor design decisions. In *Proceedings of the 1994 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 147–152, Apr. 1994.

[8] D. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin–Madison, June 1997.

[9] B. Calder and D. Grunwald. Next cache line and set prediction. In *Proceedings of the International Symposium on Computer Architecture*, pages 287–296, Nov. 1995.

[10] B. Calder, D. Grunwald, and J. Emer. Predictive sequential associative cache. In *Proceedings of the Second IEEE Symposium on High-Performance Computer Architecture*, Feb. 1996.

[11] J. H. Edmondson and et al. Internal organization of the Alpha 21164, a 300-MHz 64-bit quad-issue CMOS RISC microprocessor. *Digital Technical Journal*, 7(1), 1995.

[12] M. Gowan, L. Biro, and D. Jackson. Power considerations in the design of the alpha 21264 microprocessor. In *35th Design Automation Conference*, 1998.

[13] K. Inoue, T. Ishihara, and K. Murakami. Way-predicting set-associative cache for high performance and low energy consumption. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 273–275, Aug. 1999.

[14] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce leakage power. In *Proceedings of the 27th International Symposium on Computer Architecture (ISCA)*, July 2001.

[15] J. Kin, M. Gupta, and W. H. Mangione-Smith. The filter cache: An energy efficient memory structure. In *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 30)*, pages 184–193, Dec. 1997.

[16] S. Manne, A. Klauser, and D. Grunwald. Pipline gating: Speculation control for energy reduction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 132–141, June 1998.

[17] C.-L. Su and A. M. Despain. Cache design trade-offs for power and performance optimization: A case study. In *Proceedings of the 1995 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 63–68, 1995.

[18] S. J. E. Wilson and N. P. Jouppi. An enhanced access and cycle time model for on-chip caches. Technical Report 93/5, Digital Equipment Corporation, Western Research Laboratory, July 1994.

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