1

Improving Power Efficiency with an Asymmetric Set-Associative Cache

Zhigang Hu[1], Stefanos Kaxiras[2], and Margaret Martonosi[3]

[1] T.J. Watson Research Center, IBM Corporation, Armonk, NY, USA
zhigangh@us.ibm.com
[2] Commnunication System and Software, Agere Systems, Allentown, PA, USA
kaxiras@agere.com
[3] Department of Electrical Engineering, Princeton University, Princeton, NJ, USA
mrm@ee.princeton.edu

Summary. Data caches are widely used in general-purpose processors as a means to hide long memory latencies. Set-associativity in these caches helps programs avoid performance problems due to cache mapping conflicts. Current set associative caches are symmetric in the sense that each way has the same number of cache lines. Moreover, each way is searched in parallel so energy is consumed by all the ways even though at most one way will hit. With this in mind, this paper proposes an asymmetric cache structure in which the size of each way can be different. The ways of the cache are different powers of two, and allow for a "tree-structured" cache in which extra associativity can be shared. We accomplish this by having two cache blocks from the large ways align with individual cache blocks in the smaller ways. This structure achieves performance comparable to a conventional cache of similar size and equal associativity. Most notably, the asymmetric cache has the nice property that accesses hit in the smaller ways can immediately terminate accesses to larger ways so that power can be saved. For the SPEC2000 benchmarks, we found cache energy per access was reduced by as much as 23% on average. The characteristics of the asymmetric set-associative design (low power, uncompromised performance, compact layout) make them particularly attractive for low power processors.

1.1 Introduction

To bridge the widening speed gap between the processor and the main memory, caches are widely employed in current general purpose microprocessors. For example, Alpha 21264 processor [8] has a 64KB, 2-way set associative L1 data cache and an L1 instruction cache of the same size. The design of caches must take into consideration many factors including hit latency, miss rate, chip area and power consumption. Balancing all these factors results in complex cache designs with multiple cache levels.

A key design choice for caches is the associativity. The associativity of a cache is the number of places in the cache where a block may reside. In the simplest form, there is only one place for each data block so the associativity is 1. In this case the cache is called a direct-mapped cache. More generally, in a n-way set associative cache, a data block could appears in n different places so to search for a match all these n locations need to be checked. Figure 1.1 shows

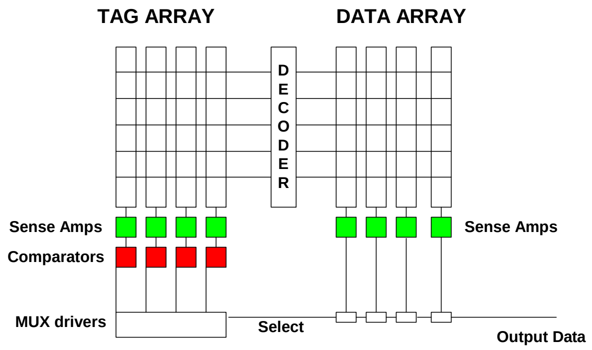

Fig. 1.1. The conventional cache organization

a typical cache organization with associativity of 4. Increasing associativity can improve cache hit rate by reducing the mapping conflict between cache lines that interference with each other. Hill and Smith [9] report that using a two-way set associative cache reduces the number of cache misses by about 30% compared to a direct-mapped cache. However, with conventional set-associative cache design, increasing associativity comes with a high cost of extended hit latency and extra power consumption.

Hit Latency: The latency of a set-associative cache is larger than that of a direct-mapped cache due to the delay associated with the multiplexing logic that selects the correct way. Based on the CACTI 3.0 [23] tool, Figure 1.2 gives the access latencies of three 16K-byte caches with different associativity. When going from a direct-mapped cache to a 2-way set associative cache, the cache latency is increased by about 25%. For many processors cache latency is on the critical path, so increasing associativity could severely impact the cycle time. Furthermore, since memory reference instructions comprise about 1/3 of total executed instructions, increasing cache latencies could result in longer execution time which will have a global impact on the performance and the power consumption of the whole microprocessor.

Power Consumption: In a conventional set-associative cache, all the ways are searched in parallel. Since at most only one way will hit, power consumed by other ways is expended without providing any useful data. From Figure 1.2, we noticed that the average energy consumed per cache access to a 2-way 16K-byte cache is about 51% higher than that of a direct-mapped cache. Increasing the associativity to 4 causes the access energy to increase by another 64%.

A major reason for the problems above is the symmetric organization and operation of conventional set associative caches. In this organization, each way is physically designed to have the same number of cache lines. Furthermore, each way is activated simultaneously thus their results come out at the same time and selection hardware must be employed to decide which result to use. In this paper, we propose a new cache architecture, called an "asymmetric cache", which allows non-uniform sizes for each cache way.

In an asymmetric cache, we propose to use larger sizes for some ways while smaller sizes for other ways. For instance, a 15K-byte 4-way set-associative cache can have 4 ways, with sizes of 8K-byte, 4K-byte, 2K-byte and 1K-byte respectively. We describe how an access to this cache can be conducted and

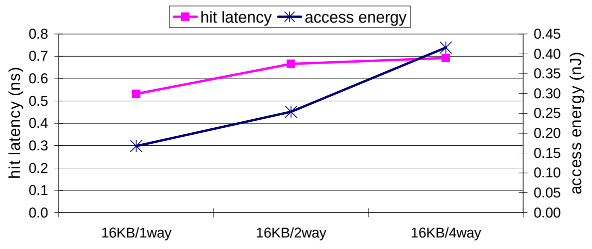

Fig. 1.2. The hit latency and energy per access for 16K-byte caches with different associativity. From left to right: direct-mapped, 2-way set associative and 4-way set associative.

how LRU replacement policy can be implemented in such a cache. We show that asymmetric caches have similar performance compared to conventional caches of similar sizes and associativity. Furthermore, since in asymmetric caches, smaller ways are faster, we show how a hit on smaller ways can immediately signal other larger ways to stop the lookup. This effect, similar to "Short Circuit Evaluation" of Boolean expressions, can reduce the average power consumed by the slower and larger ways. With this technique, the asymmetric cache described above achieves up to 23% cache energy savings compared to a 4-way conventional cache of similar size.

The structure of the paper is as follows. In Section 2, we discuss related work and in Section 3 we explain the simulation environment and machine model used to evaluate our proposed structure. Next in Section 4, we introduce in detail the structure of an asymmetric set-associative cache and discuss some advantages of this structure. In Section 5, we demonstrate our simulation results. Section 6 compares asymmetric caches with other design options and discusses our plans for future work. Finally Section 7 concludes the paper.

1.2 Related work

Caches have been the subject of much research. In general there are two main research categories. One category of research focuses on the internal structure and address mapping design within a single cache. Group associative caches [18] and DASC (Direct-mapped Access Set-associative Check) caches [22] are examples in this category. Both try to achieve the miss rate of a set associative cache with the hit latency of a direct-mapped cache by combining an associative tag array with a direct-mapped data array. In a group-associative cache, a direct-mapped cache is dynamically partitioned into groups of cache lines. Each group functions as a set as in a conventional set-associative cache. Each memory block can map to any position within a group instead of a single position in a conventional direct-mapped cache. The exact position of this block is recorded in a directory which is accessed in parallel with data/tag array. In DASC caches [22], the tag array is n-way set-associative but the data array is direct-mapped. For each memory request, data in the privileged location is optimistically used. If the tag check indicates a miss on the privileged location, all activities using the speculative data must be canceled. Since the tag array is set-associative, a hit on alternative locations can also be determined during the tag check. On a miss to all the alternative locations, the referenced data must be served from next level of the memory hierarchy. Other work in this category includes column-associative cache [1], skewed associativity cache [2] and the difference-bit cache [13].

Another category of cache research tries to split the data cache into typically two sub-caches to capture different memory access patterns. Examples in this category include —among others—split temporal spatial data caches (STS) [17], split spatial/non-spatial caches [19], victim buffers [12] and filter caches [11]. A survey of this category of research can be found in [20].

Our work is similar to the skewed associativity work [2] in that each way is indexed differently. However, in asymmetric caches, the difference in indexing stems from the different size of each cache way but not by the deliberate use of different decoders for each way. Within each way, we retain the conventional index function to avoid adding new decoders. Since our work is focused on power consumption instead of miss rate, these two mechanisms are actually orthogonal to each other. It is possible to combine the two to achieve different trade-offs between power and performance.

Beyond research focusing on miss rate there is also research focusing on latency and power consumption. In an n-way set-associative cache, each cache line has n possible locations to be placed so the possibility that a useful cache line is driven out due to mapping conflict is reduced. However, compared to direct-mapped caches, conventional set associative caches incur longer delay and higher power consumption for each cache access. These two problems have been the focus of much research [4, 5, 15, 27]. The general solution is to assign priorities either dynamically or statically to the possible locations. For each cache access, the location with the highest priority is first checked for a hit. If it is a hit, then the access is completed immediately without looking at other locations. If it is a miss, however, the remaining candidate locations are checked subsequently. This technique has been employed in some commercial processors [24, 26]. In the rest of this section, we describe two previous proposals to improve the energy inefficiency of set-associative caches.

1.2.1 Phased Cache

In a phased cache, accesses to tags and data are serialized, as shown in Figure 1.3(b). During the first stage, all the tags in the selected set are examined in parallel. The data array is not touched in this step. If the tag comparison indicates that there is a hit, then the data array in the hit way is accessed during the second stage. Otherwise, the second stage is skipped and the next level cache is accessed.

Phased cache can greatly reduce the energy consumed by each access since the tag array consumes much less energy than the data array. Also, only at most one way in the data array is activated for each cache access. However, in the common situation where there is a hit, the cache access latency is increased which may delay the whole processor and lead to more energy consumed in other parts of the processor. Therefore, an evaluation of the phased cache must take the complete processor into account.

1.2.2 Way Prediction Cache

In a way prediction cache, as shown in Figure 1.3(c), one cache way is speculatively chosen by a predictor to be accessed before the other ways. If a hit is detected in the predicted way, the access latency and power consumption are similar to that of a direct-mapped cache. On the other hand, if a miss is detected in the predicted way, the remaining ways are accessed in parallel as

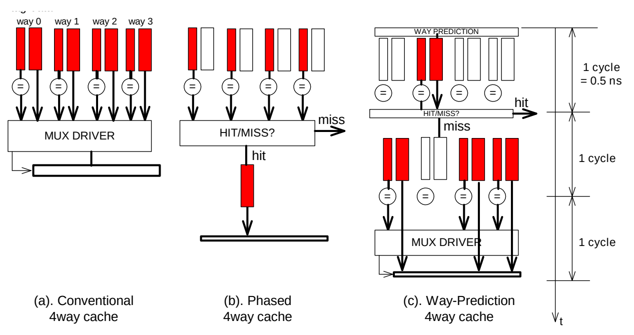

Fig. 1.3. Access procedure for different cache structures

in the conventional caches. A simple MRU (Most Recently Used) [4, 5, 15, 10] prediction policy is typically employed to exploit cache access locality. The effectiveness of way prediction cache largely depends on the accuracy of the way prediction. Our simulations reveal that the simple MRU algorithms have a prediction rate of about 84%, which is in accordance with results in [10]. The way prediction hardware itself may, however, incur some extra latency and energy consumption.

Both phased caches and way prediction caches maintain the symmetric structure of conventional caches, but they modify the access procedure to trade extra latency for reduced power consumption. In the paper, we propose a structurally asymmetric cache design which achieves a different trade-off between latency, power consumption and design complexity. Before discuss this structure in detail, we first introduce the simulator and the benchmarks we used to evaluate our design.

1.3 Methodology and Modeling

Our performance results shown in this paper are based on simulations using the SimpleScalar [3] tool set. The cache energy and timing results are obtained with CACTI 3.0 [23] which is an updated version of CACTI [25]. We augmented the cache model in SimpleScalar to simulate asymmetric caches, phased caches and way prediction caches. Our simulated processor is a 2GHz 4-issue out-of-order processor based on the 0.1um technology. The main parameters of this processor are shown in Table 1.1.

We evaluate our results using benchmarks from the SPEC CPU2000 benchmark suite [6]. The benchmarks are compiled and statically linked for the Alpha instruction set using the Compaq Alpha compiler with SPEC *peak* settings. For each program, we skip the first 1 billion instructions to avoid the initial startup behavior of the benchmarks. We then simulate the program until 2 billion instructions are committed. Our simulation is conducted with SimpleScalar's EIO traces using the reference input set to ensure reproducible results for each benchmark across multiple simulations.

1.4 Asymmetric Set Associative Cache

In this section, we will describe the basic structure, the access policy and replacement policy of asymmetric caches and analyze the latency and power consumption associated with this design.

General	
Clock Frequency	2GHz (0.5ns cycle time)
Feature Size	0.1um
Processor Core	
Instruction Window	64-RUU, 32-LSQ
Issue width	4 instructions per cycle
Functional Units	4 IntALU,1 IntMult/Div, 4 FPALU,1 FPMult/Div, 2 MemPorts
Memory Hierarchy	
L1 Dcache Size	16KB, 4-way, 32B blocks, 2-cycle
L1 Icache Size	8KB, 1-way, 32B blocks, 1-cycle
L2	Unified, 512KB, 8-way LRU, 64B blocks,8-cycle latency, WB
Memory	100 cycles
TLB Size	128-entry, 30-cycle miss penalty

Table 1.1. Configuration of Simulated Processor

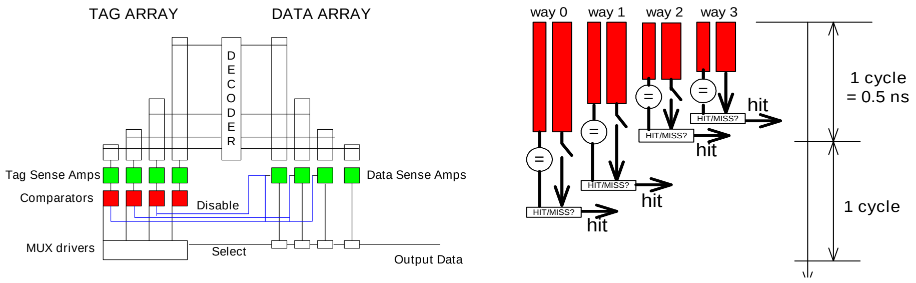

Fig. 1.4. Structure (left) and access procedure (right) of an asymmetric cache.

1.4.1 Structure

As the name implies, in an asymmetric set associative cache, the cache ways are asymmetric; that is, they are of different sizes. Figure 1.4 shows the diagram of the asymmetric 4-way set associative cache that is modeled in our simulation. In a 15K-byte asymmetric cache with 32-byte cache lines, each of the four ways has 256, 128, 64 and 32 cache lines respectively.

Since the sizes of each way are different, decoder design becomes an issue for asymmetric caches. In conventional cache design, caches are broken into several smaller banks/blocks to balance the wire length of each direction [23, 25]. With this design, the large decoder shown in Figure 1.1 is actually made up of simpler sub-decoders. By choosing sizes of each way to be powers of two, we can share these subdecoders among the ways. Thus, no extra decoders are required in asymmetric caches.

Due to the size difference of each way, asymmetric caches have the following characteristics:

1. Because of their smaller sizes, smaller ways are faster.
2. Tag comparisons in different ways happen in different speeds: hits are detected faster in smaller ways.
3. Smaller ways consume less power.

In the following subsections we describe how an asymmetric cache is accessed and how the replacement is conducted. We will also demonstrate how

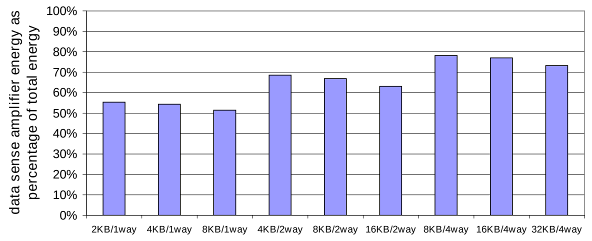

Fig. 1.5. The energy of data sense amplifiers as a percentage of total energy per cache access for various cache configurations.

we can take advantage of the asymmetric characteristics to achieve a new trade-off between performance and power consumption.

1.4.2 Access Policy

Figure 1.4 also depicts the access procedure of the asymmetric 4-way set-associative cache. Similarly to conventional set associative caches, each way in asymmetric caches starts searching in parallel to look for a hit. Unlike conventional caches however, asymmetric caches have different hit latencies for each way due to their size differences. In the situation when an access hits a smaller (thus faster) way, it is desirable to signal other slower ways to terminate their lookup. We refer to this mechanism as “shorting the lookups”. This can reduce cache hit latency and power consumption.

Terminating a lookup can be conducted at various stages in the process of a cache access. We can add termination control logic to the wordline, the bitline, the sense amplifier and the output drivers. The choice of the control point depends on the latency overhead and the power savings achieved. Figure 1.5 shows the percentage of cache energy per access attributed to data sense amplifiers for various caches with 32-byte cache line size in 0.1um technology. The simulation are conducted with the CACTI 3.0 tool [23]. Across the different cache configurations, we observed that more than half of the energy per access is consumed on the data sense amplifiers. This suggests that to save power we should gate the data sense amplifiers and only activate them when needed.

In [23], control logic is proposed to gate the foot transistors of the sense amplifiers. In this section, we introduce an alternative way to gate the sense amplifiers. As shown in Figure 1.6, typical sense amplifier designs incorporate a *sense* signal that is pulled down (simultaneously with wordline) to engage the sense amplifier [16]. Our approach is to gate the sense signal of the larger ways using the result (miss) from the smaller ways. Only in case of a miss in the smaller ways the sense amplifiers are enabled in the larger ways. We can cascade this gating signal from smaller to larger ways but we may delay a hit on the largest way. Alternatively any hit in any of the ways disables sensing in all larger ways. We simulate the latter in our evaluations.

Gating the sense signal for a small period does not affect the correctness of the circuits as long as we are dealing with static RAM cells [7]. In this case, the static memory cell only enlarges the differential voltage in the bit and bit-bar lines making it easier for the sense amplifier to amplify this to full swing down the road. The same is not true for dynamic RAM cells, however. There the sense signal has to be asserted simultaneously with wordline signal

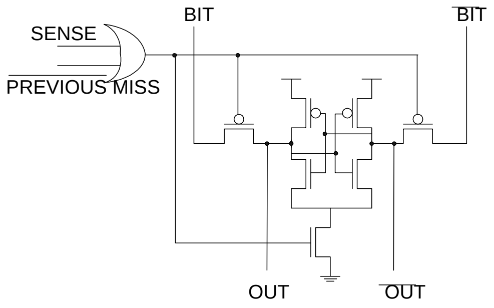

Fig. 1.6. Conventional sense amplifier augmented with gated sense signal.

since with the passage of time it becomes harder for the sense amps to detect the differential between bit and bit-bar. Under such conditions, transient noise can easily introduce errors.

Gating the data sense amplifiers will affect the hit latency and the power consumption of a cache access. We discuss this issue in detail below:

Hit Latency: Since the larger ways only activate their sense amplifiers after the tag comparison completed in all smaller ways, access latencies to these ways could be extended. For the 15K-byte asymmetric 4-way cache we simulated, our evaluation based on CACTI 3 shows that while the access to the two smaller ways (way 2 and way 3 in Figure 1.4) can be finished in a single clock cycle, the access to the two larger ways (way 0 and way 1 in Figure 1.4) needs two cycles to complete. Compared to a conventional 4-way cache, the hit latency to the smaller ways has a one-cycle advantage.

Power Consumption: The potential hit latency benefits can lead to improvements in a program's overall energy consumption and energy-delay product since they may reduce the program execution time. Moreover, our scheme for shorting cache lookups can be effective in reducing cache energy per hit, when hits occur in the smaller cache ways. As was explained above, if an access hits on a smaller way, the larger ways can be prevented from continuing the lookups. Thus, the energy consumed by the data sense amplifiers can be saved. Thus, early hits on faster ways are much more power-efficient than other hits. As shown in Section 4, this significantly reduces the total power consumption of asymmetric caches.

1.4.3 Replacement Policy

Because of the asymmetry among cache ways, conventional replacement algorithms are not directly applicable to an asymmetric cache. Replacement strategies in an asymmetric cache can affect where heavily accessed data is stored in the cache, thus impacting both performance and power consumption.

Since hits in a smaller way have smaller latency and lower power consumption, one might want to devise schemes in which the most heavily accessed data is dynamically pushed to the smallest ways. However, data movement within the cache would significantly increase the power consumption. In this paper we examine asymmetric caches using simple LRU replacement policies without data movement.

An LRU replacement strategy is necessary to maintain low miss rates in set associative caches, especially when power is a major concern and we want

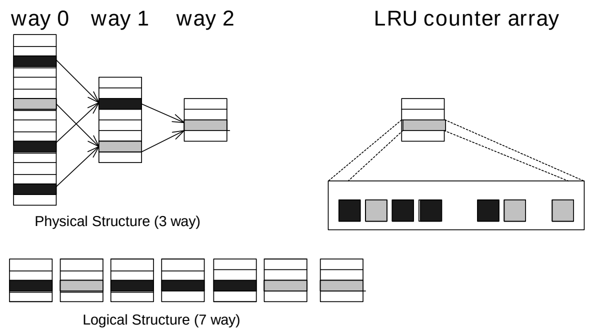

Fig. 1.7. An LRU counter based replacement policy for asymmetric cache.

to minimize accesses to the outlying levels of the memory hierarchy. In asymmetric caches, LRU replacement is slightly more complex than in conventional caches. We illustrate this assuming an LRU implementation with N modulo-N counters per set, where N is the associativity. The counters are updated so as to preserve a total order within the set, the largest value indicating the LRU line in the set. Figure 1.7 shows the LRU scheme of a 3-way asymmetric cache. In the asymmetric cache, for the purpose of LRU replacement, we consider that the number of sets to be equal to the number of lines in the smallest way. This is illustrated by the "logical structure" of the asymmetric cache shown in Figure 1.7. Each set however, consists of more lines than the associativity of the cache. Specifically each set contains all the alternative lines that map to the same line in the smallest way. Thus, the LRU array has only as many entries as the size of the smallest way but each entry contains more counters for the larger ways. Assume the asymmetric cache in the figure has 256,128 and 64 cache lines for each way respectively. Then there are just 64 LRU counter entries corresponding to the 64 lines of the smallest way. Each LRU counter entry, as shown on the right side of the figure, contains 4 LRU counters for the largest way (256 lines), 2 counters for the second largest (128 lines), and 1 counter for the smallest way (64 lines). When new data items are brought into the cache, a set of lines is selected according to the address of the new data. This address maps onto a single line in each of the 3 ways specifying a unique path in the mapping tree (see Figure 1.7). Only the LRU counters that correspond to the specific mapping path are considered for the replacement decision. The evicted line is the one with the largest value among the counters selected. However, the LRU entry is updated with any access that corresponds to the mapping tree, and therefore behaves as if it were not 3-way but 7-way set associative. Overall, the LRU counters in a 3-way asymmetric cache are maintained like a 7-way conventional set-associative cache, but the victim selection is done similarly to a 3-way conventional cache.

While the above paragraph shows that the counter based LRU replacement algorithm can be revised to work with asymmetric caches, it complicates the update and lookup operations of the LRU counters. Here we introduce an alternative way to implement a decay based replacement algorithm that emulates a true LRU algorithm. Cache decay [14] uses 2-bit counters to gauge the idle time of cache lines and proposes to shut off cache lines with long idle times to save cache leakage energy. These decay counters contain similar information as LRU counters. Specifically, within a cache set, the cache line with the

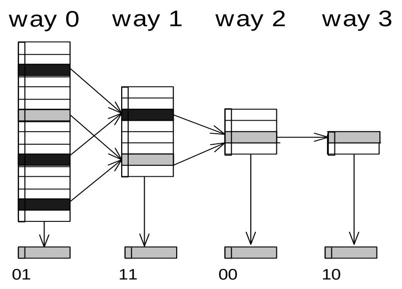

Fig. 1.8. A decay counter based replacement policy for asymmetric cache. The cache line with the longest idle time (i.e. largest decay counter value) is replaced.

largest idle time is the LRU block and conversely, the one with the shortest idle time is the MRU block. This observation suggests that we can use the decay counters to approximate LRU counters. For each cache replacement, we evict the cache block with the largest decay counter value within the set. Compared to the LRU implementation described in the previous paragraph, the update and lookup operations for the decay counters are much more simpler. As in cache decay, we reset the decay counter with every cache line access and increase the counter with every global clock tick. During an eviction, all decay counters within a cache set are read out and the cache line with the largest counter value is chosen for replacement. In our simulation, we use a 2-bit local counter for each cache line and use a global cycle counter which ticks every 256 cycles. With these coarse grained counters, an interesting situation could occur when more than one decay counter holds the same largest value within a cache set. In this situation, we consider all those cache lines with the largest counter value as candidates for replacement. Among these candidates, there is no further information to decide their exact LRU order, so we can assign priority based on their way sizes. Since accesses to smaller ways are faster and more energy-efficient, we prefer to place active cache lines in smaller ways. This leads to our first policy, where we always choose the smaller way to replace in case of equal counters. We call this policy "favor smaller ways". Conversely, the "favor larger ways" policy tries to replace the larger way if the counters are the same. In comparison, we also considered an ideal implementation where the counter is very fine grained so that the situation where two decay counters equal never occurs. We call this policy "ideal LRU". Notice that the counter-based LRU algorithm shown in Figure 1.7 can be used to implement "ideal LRU". In the next section, we will evaluate the performance and power consumption of these three policies.

1.5 Results

In this section, we will examine simulation results for asymmetric caches compared to conventional symmetric caches. The conventional 4-way set associative cache has size of 16K-byte and block size of 32 bytes, with each way having 128 cache lines. The 15K-byte asymmetric caches have 256, 128, 64 and 32 cache lines for its 4 ways respectively. For the conventional cache, we employ a true LRU algorithm. The asymmetric caches are evaluated with the three LRU algorithms we described in the previous section, including "favor smaller ways", "favor larger ways" and "ideal LRU".

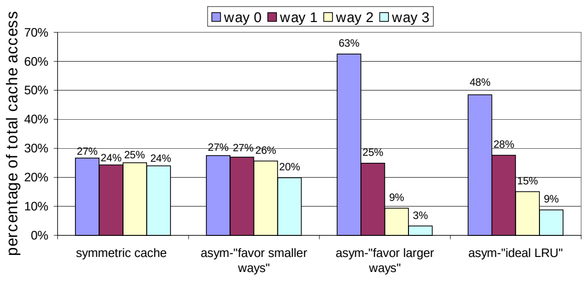

Fig. 1.9. Access to each way with a conventional symmetric cache and 3 asymmetric caches with different LRU implementations.

1.5.1 Access Frequency to Each Way

Different flavors of LRU implementation lead to different distributions of cache accesses in each way. This effect is illustrated in Figure 1.9. As expected, in a conventional symmetric cache, the 4 ways are roughly equally exercised so each way has about 25% probability of being accessed. In asymmetric caches, with the "ideal LRU" algorithm, the access distribution to each way matches their relative sizes. In our simulated asymmetric caches, the distribution follows the "8:4:2:1" size ratio. More interesting distributions can be observed with the decay counter-based LRU implementations. When the algorithm favors smaller ways, these ways received many more accesses than their sizes would suggest. Particularly, in the sample configuration shown in this experiment, with 256-cycle global counter and 2-bit local counters, the "favor smaller ways" LRU implementation achieved roughly equal distributions of accesses to each way, even though the size of the smallest way is only 1/8 of the largest way. Conversely, when the LRU algorithm favors larger ways, smaller ways are very infrequently accessed and the bulk of the accesses are directed to larger ways.

In the next 2 subsections, we will explore the effect of access policy and replacement policy on the performance and power consumption of asymmetric caches.

1.5.2 Performance

Figure 1.10 depicts the IPC for SPEC2000 benchmarks with conventional vs. asymmetric 4-way 16K-byte caches. In asymmetric caches, hits on smaller ways have lower latencies than larger ways. However, the impact of this effect on performance is not significant for two reasons: the aggressive out-of-order execution and the fully pipelined data cache access. In [8], it has been estimated that increasing cache latency by 1 cycle degrades overall performance by about 4% for the Alpha 21264 microprocessor. We expect this effect to be even smaller for asymmetric caches since accesses hit smaller ways only part of the time. As shown in the figure, the average IPC difference is less than 1% comparing a 16K-byte conventional symmetric cache and a 15K-byte asymmetric cache with various LRU implementations. In benchmarks such as vpr and apsi, the cache size difference (15K vs.16K) causes some recognizable IPC degradation while in most other benchmarks, this effect is not significant. In many benchmarks, such as gzip, gap, wupwise and applu, the IPC of "favor smaller way" outperforms other policies because more accesses hit on

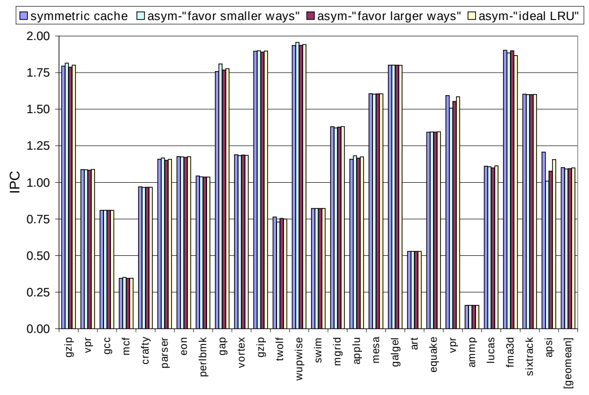

Fig. 1.10. IPC of asymmetric vs. conventional 4-way caches for SPEC2000

	Way 0 256 lines	Way 1 128 lines	Way 2 64 lines	Way 3 32 lines	Overall
(Favor smaller ways)					
Access frequency(%)	0.27	0.27	0.26	0.20	
Access energy	0.27	0.22	0.17	0.10	0.77
(Favor larger ways)					
Access frequency(%)	0.63	0.25	0.09	0.03	
Access energy	0.63	0.21	0.06	0.02	0.91
(Ideal LRU)					
Access frequency(%)	0.48	0.28	0.15	0.09	
Access energy	0.48	0.23	0.10	0.04	0.86

Table 1.2. Access frequency and per-access energy for asymmetric caches

the smaller, faster ways. Overall, the performance is roughly stable among all the configurations because all the caches considered have similar size and all use an LRU replacement policy.

1.5.3 Power Consumption

This section compares the power consumption of our asymmetric caches with the similar-sized conventional 4-way cache. Since both caches are 4-way set associative, we assume the energy consumed by the mux drivers and the output drivers are similar (See Figure 1.1 and Figure 1.4, these drivers account for 1% - 2% energy in a 4-way cache.) Thus, we exclude them from our comparison. To calculate access energy for each way, we utilized CACTI 3.0 [23] assuming a 0.1um technology. Considering the "shorting the lookups" effect, we estimated that hitting in the four ways (from the smallest to the largest way) in our sample asymmetric cache consumes about 50%, 66%, 83% and 100% of the access energy of a conventional cache.

We estimate the energy per access of a conventional 4-way cache (excluding the mux drivers and output drives) as sum of the access energy to each way. For the asymmetric cache, by applying lookup-shorting, we avoid the data sense amplifier energy in the larger cache ways if we determine in time that we have a hit in one of the smaller ways. Considering this effect, we estimate the energy saving of each way as $prob(access\_in\_this\_way) * energy\_for\_hit\_on\_this\_way$. Based on this estimation, Table 1.2 estimates average cache access energy considering the access frequency to each cache way. Table 1.2 shows that compared to the conventional cache, an asymmetric cache achieves about

	Caches	Conven.	Phased	WP	Asym
hit	Latency(cycles)	2	3	1/3	1/1/2/2
	Power	1	0.37	0.25/1	0.5/0.66/0.83/1
miss	Latency(cycles)	2	2	3	2
	Power	1	0.14	1	1

Table 1.3. Comparison of power consumption and latency for asymmetric caches vs. conventional caches, phased caches and way prediction caches. Note: Power is normalized to the conventional caches.

23%, 9% and 14% energy savings with "favor smaller ways", "favor larger ways" and "ideal LRU" replacement policies respectively.

1.6 Discussion and Future Work

In this section we compare asymmetric caches with phased caches and way prediction caches and discuss some design issues associated with asymmetric caches. We also outline some directions for future work.

1.6.1 Comparing Asymmetric Caches to Previous Proposals

Asymmetric caches, like phased caches and way prediction caches, aim to achieve a new trade-off between latency, power consumption and design complexity. Table 1.3 presents a detailed comparison of latency and power consumption for these cache schemes. Notice that for way prediction caches, there are two situations for a cache hit: hit on the predicted way or hit on other ways. For 4-way set-associative asymmetric caches, there are four situations for a cache hit: hit on each of the four ways respectively. These situations incur different latency/power consumption as shown in the table.

Phased Caches: In phased caches, the lookup of tag array and data array is serialized: the data array is accessed only when the tag comparison indicates a hit. As indicated by Table 1.3, phased cache utilized less cache access power no matter hit or miss. However, in the common case when there is a cache hit, the latency is one cycle longer than a conventional cache. This extra cycle latency is tolerable in L2 caches but is a rather big overhead for L1 caches which are typically performance constrained.

Way Prediction Caches: In way prediction caches, the lookup of one special way (typically the MRU way) is given priority over other ways. If it turns out that this way gets hit, then both the latency and the power consumption are improved. As in other predictive mechanisms, the effectiveness of way prediction caches are highly dependent on the accuracy of way prediction and the associated access pattern. If for some reason the access pattern is changed, for example, under a SMT environment, then the accuracy of way prediction might deteriorate and the extended latency associated with wrong way prediction could harm the overall performance.

Both phased caches and way prediction caches try to explicitly serialize the cache access process. On the contrary, asymmetric caches achieved implicit serialization through the size/speed differences among the ways. It is tempting but difficult to reach a simple conclusion about which scheme is the best, because each scheme targets at different balance and trade-off among many design factors such as latency, power consumption and complexity. Together these schemes form a rich set of design choices for cache designers. In the next subsection we discuss some design issues related to the evaluation of asymmetric caches vs. other cache structures.

1.6.2 Design Issues

Instruction Scheduling: Load/Store instructions bring challenges to the instruction scheduler because their latencies are variable and unknown at the time of scheduling. In particular, the latency for cache misses is much longer than that for hits. Since cache hits are typically more common than cache misses, current processors often optimistically issue a load-dependent instruction assuming the load will hit in the cache [21]. If it turns out that the load misses in the cache, the speculatively-issued instructions are squashed and re-issued after the load is completed. With way prediction and asymmetric caches, this strategy is further complicated because the hit latency depends on where the hit occurs. In this paper, we assume a somewhat ideal scheduler which knows the exact cache access latency at the time of scheduling. Interesting future work would be to model a more realistic scheduler and evaluate its impact on the the cache structures we discussed in this paper, including phased caches, way-prediction caches and asymmetric caches.

Cycle Time Considerations: In the race to better performance and higher clock frequency, it can be very difficult to design a reasonably-sized cache for a given cycle time. Instead, current L1 data caches are typically pipelined into 2 or more cycles. By allowing different latencies for each way, the asymmetric cache structure provides more flexibility for fitting the cache access time into a desired latency.

1.6.3 Future Work

The idea of assigning different sizes to each way can be explored beyond achieving power savings. In this section we briefly discuss the possibility of achieving area savings using this idea. Because of the rule of locality, the most recently used way (MRU way) is more likely to be re-accessed than the least recently used way (LRU way). This phenomenon has been explored by way prediction [10]. With an asymmetric cache structure, we can explore this phenomenon by moving data around so that the MRU line in a set is always placed in the largest way. Similarly, we can place the LRU line always in the smallest way. Thus at the cost of extra data migrations among the ways, an asymmetric cache could possibly achieve the miss rate of a much larger conventional cache. This scheme is especially suitable for L2 caches since the area savings there is more significant while extra data movements could be better tolerated. In this paper we did not explore this mechanism but it could be a promising direction for future work.

1.7 Conclusions

Current set-associative caches are physically symmetric. That is, all the ways have the same number of cache lines. Moreover, current set-associative caches are operated symmetrically which means all the ways are looked up in parallel. This results in both extended hit latency and increased power consumption compared to a direct-mapped cache. In this paper we take a different approach: we propose set-associative caches in which different ways have different sizes. In these asymmetric caches, sizes of different ways are different powers-of-2 and allow for a "tree-structured" cache. Extra associativity is shared by having two cache blocks from the large ways align with individual cache blocks in the smaller ways. An LRU replacement policy can be implemented by treating all the items in a “mapping tree” as a single set with higher associativity. Replacement decisions take into account only the items that correspond to a single path within the mapping tree. An alternative LRU replacement is to use decay counters instead of LRU counters. The advantage of this implementation is that decay counters are easier to maintain for asymmetric caches.

Because of their different size, cache ways in asymmetric caches have different access times and power characteristics. In particular, smaller ways can be accessed faster and at the same time expend less energy. We can further exploit a hit on a fast cache way by “shorting” the lookups in the slower cache ways.

Thus, asymmetric caches have the benefit of lower power consumption in the smaller ways while maintaining the performance of conventional caches. By immediately terminating lookups in larger ways when detecting a hit on smaller ways, the average cache access energy is reduced by as much as 23% for SPEC2000.

Asymmetric cache architectures do not require any elaborate new hardware but rather they are simple variations in the geometry of conventional set-associative caches. This minimal-cost approach results in power savings and with further optimizations could even provide higher performance at the same time.

References

1. Agarwal, A and Pudar, S (1992) Column-associative caches: A technique for reducing the miss rate of direct-mapped caches. In: Proceedings of the 20th Annual International Symposium on Computer Architecture
2. Bodin, F and Seznec, A (1995) Skewed associativity enhances performance predictability. In: Proceedings of the 22nd Annual International Symposium on Computer Architecture
3. Burger D, Austin T, and Bennett S, (1996) Evaluating future microprocessors: the SimpleScalar tool set. In: Tech. Report TR-1308, Univ. of Wisconsin-Madison Computer Sciences Dept.
4. Calder B, Grunwald D, and Emer J (1996) Predictive sequential associative cache. In: Proceedings of the 2nd Annual International Symposium on High Performance Computer Architecture
5. Chang J. H., Chao J., and So K. (1987) Cache design of a sub-micron CMOS system370. In: Proceedings of the 14th Annual International Symposium on Computer Architecture
6. SPEC Corporation (2000) WWW Site http://www.spec.org.
7. Diodato P (2001) Personal communication.
8. Gwennap, L (1996) Digital 21264 sets new standard. In: Microprocessor Report, October 1996, pp. 11–16
9. Hill M, and Smith A (1989) Evaluating associativity in CPU caches. In: IEEE Transactions on Computers (38)12: pp. 1612–1630
10. Inoue K, Ishihara T, and Murakami K (1999) Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption. In: Proceedings of the 1999 International Symposium on Low Power Electronics and Design
11. Johnson M, and Mangione-Smith, W. (1997) The filter cache: An energy efficient memory structure. In: Proceedings of the 30th Annual International Symposium on Microarchitecture
12. Jouppi, N (1990) Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. In: Proceedings of the 17th Annual International Symposium on Computer Architecture
13. Juan T, Lang T, and Navarro J (1996) The difference-bit cache. In: Proceedings of the 23rd Annual International Symposium on Computer Architecture

14. Kaxiras S, Hu Z, and Martonosi M (2001) Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In: Proceedings of the 28th Annual International Symposium on Computer Architecture
15. Kessler R, Jooss R, Lebeck A, Hill M (1989) Inexpensive implementation of set-associativity. In: Proceedings of the 16th annual international symposium on Computer Architecture, pp. 131-139
16. Villa L, Zhang M, and Asanovic K (2000) Dynamic Zero Compression for Cache Energy Reduction. In: Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture
17. Milutinovic, V, Markovic B and Tremblay M (1996) The Split Temporal/Spacial Cache: Initial Performance Analysis. In: Proceedings of the SCIzzL-5
18. Peir J, Lee Y, and Hsu W (1998) Capturing Dynamic Memory Reference Behavior with Adaptive Cache Topology. In: Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems
19. Prvulovic, M, Marinov D, Dimitrijevic Z and Milutinovic, C (1999) The Split Spatial/Non-Spacial Cache: A performance and Complexity Analysis. In: IEEE TCCA Newsletters
20. Sahuquillo J, and Pont A (2000) Splitting the Data Cache: A Survey. In: IEEE Concurrency 8(3): pp. 30–35
21. Seznec A (1993) A case for two-way skewed-associative caches. In: Proceedings of the 20th Annual International Symposium on Computer Architecture, pp. 169–178.
22. Seznec A (1995) DASC cache. In: Proceedings of the 1st Annual International Symposium on High Performance Computer Architecture
23. Shivakumar P, and Jouppi N (2001) Cacti 3.0: An integrated cache timing, power, and area model. In: Technical report 2001/2, Compaq Western Research Lab
24. Tremblay M, and O'Connor J (1996) UltraSparcI: A four-issue processor supporting multimedia. In: IEEE Micro (16)2: pp. 42–50
25. Wilton S and Jouppi N (1994) An Enhanced Access and Cycle Time Model for On-chip Caches In: Research Report 1993/5, Compaq Western Research Lab
26. Yeager, K. (1996) The MIPS R10000 Superscalar Microprocessor. In: IEEE Micro (16)2: pp. 28–40
27. Zhang C, Zhang X, Yan Y (1997) Two fast and high-associativity cache schemes. In: IEEE Micro (17)5: pp. 40–49