Improving Cache Power Efficiency with an Asymmetric Set-Associative Cache

Zhigang Hu and Margaret Martonosi
Department of Electrical Engineering
Princeton University
hzg,martonosi@ee.princeton.edu

Stefanos Kaxiras
Circuits and Systems Research Lab
Agere Systems
kaxiras@agere.com

Abstract

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. Many programs, however, need high associativity for only some of their frequently-referenced addresses and tolerate much lower associativity for the remainder of the references. With this variability 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 miss rates comparable (on average 3% better for SPEC2000) to a conventional cache of the same size and 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 17% on average.

1 Introduction

To attack the speed gap between processor and main memory, aggressive cache architectures are widely employed in current general purpose microprocessors. For example, Alpha 21264 [5] processor has a 64KB, 2-way set associative L1 data cache and an L1 instruction cache of the same size. Cache design is a trade-off of many factors including hit latency, miss rate, chip area and power consumption. Balancing all these factors results in complex designs.

In this paper we propose set associative caches with asymmetrical ways. This simple technique is based on observed access behavior in set associative caches and can potentially provide benefits in three areas: hit latency, miss rate, and power consumption. Figure 1 shows a typical cache organization. In this organization, each way is symmetric. That is, each set is designed to have the same number of ways. However, during actual program execution, depending on the memory address mapping and memory access behavior, each set may have different associative requirements. For example, in an extreme case, a lot of addresses may map to a single set while no address map to another. This wastes cache space and leads to conflict misses which can be eliminated if a more efficient mapping mechanism were used. Figure 2 depicts the number of misses to each set in a 32K direct-mapped L1 data cache for a SPEC2000 program, gzip. This figure clearly demonstrates that some sets experience many more misses than others. Intuitively, we could optimize the cache behavior by assigning more ways to these sets than others.

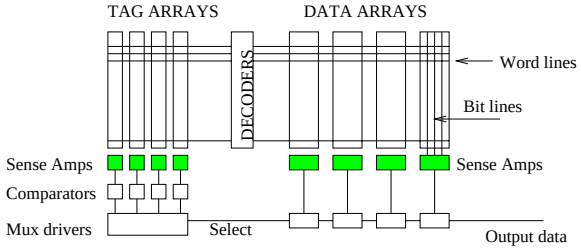

Figure 1: The conventional cache organization

Different programs may also have different set asso-

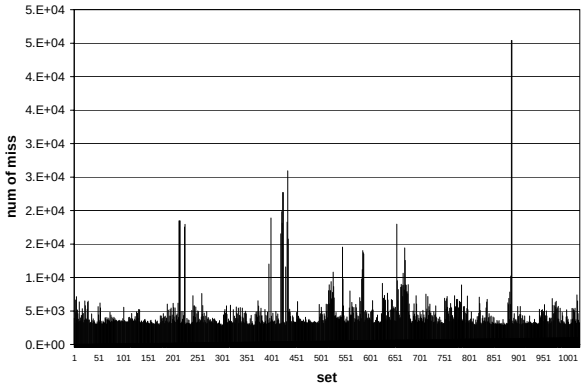

Figure 2: Misses to each set with a 32KB direct-mapped cache for gzip

ciativity requirements. Figure 3 shows the miss rate of 32KB caches with different associativity for SPEC2000 benchmarks. Most benchmarks get remarkable improvement when increasing associativity from direct-mapped to 2-way set associative. On average, a 2-way cache can reduce miss rate by 13% compared to a direct-mapped one with the same size. The improvements from 2-way to 4-way is rather small for most benchmarks. On average, a 4-way cache reduce the miss rate by only 1.5% compared to a 2-way cache. However, for some benchmarks, such as crafty, perlbmk and galgel, the miss rate improvements are quite significant. These data indicate that most benchmarks have an associativity requirement of around 2 and only some of them need more.

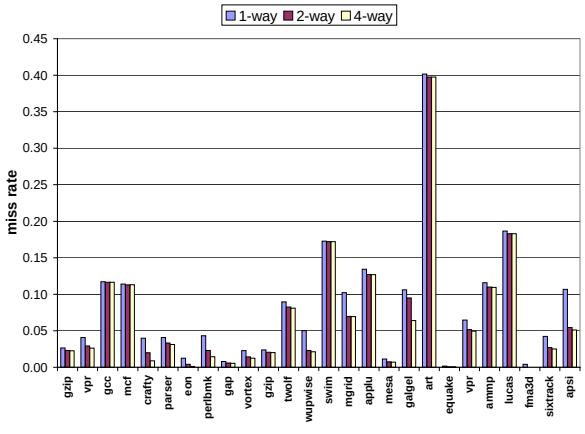

Figure 3: Miss rate of 32KB L1 data caches with different associativity for SPEC2000

1.1 Related Work

Cache organization has 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 cache [12] and DASC (Direct-mapped Access Set-associative Check) cache [17] 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 group-associative cache [12], a direct-mapped cache is dynamically partitioned into groups of cache lines. Each group functions as a set as in an conventional set-associative cache. Each memory block can map to 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 [17], 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 is set-associative, a hit on alternative locations can be also be determined during the tag check. On a miss to all the alternate 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 [8].

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—the split temporal spatial data cache (STS) [11], the split spatial/non-spatial cache [13], victim buffer [7], filter cache [6]. A survey of this category of research can be found in [15].

Our work is also 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 and not by the deliberate use of different decoders for each way. Thus, 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's possible to combine the two to achieve different trade-offs between power and performance.

1.2 Contributions

In this paper, we propose an asymmetric structure for set associative cache where the size of each way can be different. Because each set in a cache has different associativity requirements and on a higher level, different programs also have this property, we propose to use smaller sizes in higher associativity ways. For instance, a 64K 4-way set-associative cache can have 4 ways of size 1024, 512, 256 and 256 lines respectively. We show that this organization has better miss rate than the equal-capacity cache of either direct-mapped or conventional 2-way or 4-way set-associativity. Furthermore, since smaller ways are faster, we show how a hit on those ways can immediately signal other ways to stop the lookup. This effect, similar to "Short Circuit Evaluation" of boolean expressions, can reduce the average power consumed in the slower and larger ways. By applying this technique, the asymmetric cache described above achieves 17% cache energy savings compared to a 4-way conventional cache of the same size.

The structure of the paper is as follows. In Section 2, we explain the simulation environment and machine model used to evaluate our proposed structure. Next in Section 3, we introduce details about the structure of the asymmetric set-associative cache and discuss some advantages of this structure. In Section 4, we demonstrate our simulation results. Finally, Section 5 concludes the paper and discusses our plans for future work.

2 Methodology and Modeling

2.1 Simulator

Simulations in this paper are based on the SimpleScalar framework [3] and CACTI 2 [19], [14]. Our model processor has sizing parameters that closely resemble Alpha 21264 [5], but without a clustered organization. The main processor and memory hierarchy parameters are shown in Table 1.

2.2 Benchmarks

We evaluate our results using benchmarks from the SPEC CPU2000 benchmark suite [18]. The benchmarks are compiled for the Alpha instruction set using the Compaq Alpha compiler with SPEC *peak* settings. For each program, we follow the recommendations in [16], but skip a minimum of 1 billion instructions. We then simulate 500M instructions using the reference input set.

Processor Core	
Instruction Window	80-RUU, 40-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	Varied size, 32B blocks, WB
L1 Icache Size	32KB, 1-way, 32B blocks, WB
L2	Unified, 1MB, 8-way LRU, 64B blocks,6-cycle latency, WB
Memory	100 cycles
TLB Size	128-entry, 30-cycle miss penalty

Table 1: Configuration of Simulated Processor

3 Asymmetric Set Associative Cache

3.1 Structure

In an asymmetric set associative cache, as the name implies, the sets are of different sizes. Figure 4 shows an asymmetric 4-way set associative cache that is modeled in our simulations. The size of each of the four ways is 1024, 512, 256 and 256 entries respectively. We set the last set to 256 lines (instead of 128) simply because in this way the asymmetrical cache can be laid out into roughly twice the space required for its largest way. In addition this sizing choice allows direct comparison with conventional caches of equal size.

In conventional cache layouts, caches are broken into several smaller blocks to balance the wire length of each direction [19]. Consequently, the large decoder shown in Figure 1 is also made up of simpler subdecoders. By carefully choosing way size of powers of two, we can share these subdecoders among the ways in asymmetric caches.

Using this design asymmetric caches have the following characteristics:

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

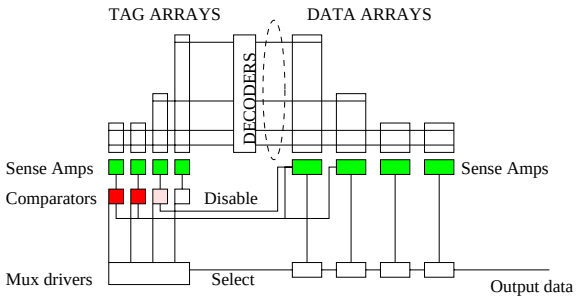

Figure 4: Structure of an asymmetric cache

In the following subsections we describe how we can take advantage of these characteristics to achieve better performance with less power.

3.2 "Shorting lookups" in Asymmetric Set Associative Caches

Similarly to conventional set associative caches, each way in asymmetric caches should be accessed in parallel to look for a hit. Unlike conventional caches however, asymmetric caches have different hit latencies depending on the size of the way that contains the correct address and data. In particular, if an address hits on a faster tag way, it is desirable to signal the slower ways to terminate their lookup. We refer to this as shorting the lookup. This behavior, similar to *short circuit evaluation*, has important effects on cache hit latency and power consumption. For this to work well, the tag comparison results of smaller ways should be known before sense amps are initiated in the larger ways. Table 2 show the latency of these two paths based on Cacti 2.0 [14]. For an 8KB 1-way cache, the time for the tag comparison to finish is 1.05916ns. This is smaller than the latency from data decoder to before data sense amps for 16KB and 32KB caches, which is 1.10357ns and 1.41125ns respectively.

Typical sense amp designs incorporate a *sense* signal that is pulled down (simultaneously with wordline) to engage the sense amp, as shown in Figure 5 [10]. Our approach is to gate the sense signal of the larger ways using the result (miss) from the smaller ways. In case of a miss in the smaller ways the sense amps are enabled in the larger ways. We can cascade this signal gating 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 former 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. [4] 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 amp 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 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.

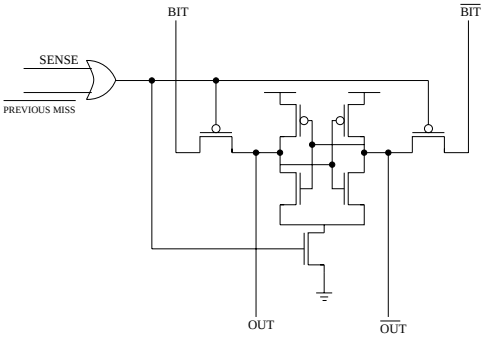

Figure 5: Conventional sense amplifier augmented with gated sense signal.

Latency	8KB 1-way	16KB 1-way	32KB 1-way
Path 1(ns)	1.05916	1.27253	1.72843
Path 2(ns)	0.82921	1.10357	1.41125

Table 2: Latencies of Path (1): From tag decoder to after tag comparator and Path (2): From data decoder to before data sense amps

3.3 Hit Latency

When a cache access hits on the faster way, data can be immediately sent back to the processor. In this case the processor does not need to wait for the results of slower ways, so the hit latency is the latency of the faster way. On the other hand, if an access misses on the faster way, the other ways have to be checked for a hit. In this situation, the hit latency is the latency of the slower ways. Therefore, the average hit latency of an asymmetric set associative cache is p(hit_on_faster_way) * smaller_latency + p(hit_on_slower_way) * larger_latency.

In current general purpose processors, an access to

L1 caches typically takes more than 1 cycle to complete [5]. Instead of waiting the whole access latency for the hit/miss results, some processors speculatively issue load-dependent instructions based on prediction of the load hit/miss [9]. Like other speculation techniques, this mechanism suffers a penalty when such a prediction turns out to be wrong. In an asymmetric cache, we are not *predicting* a way; rather, since the faster ways have a smaller latency, we learn their hit/miss results sooner. Thus, their result can be returned earlier to the processor. This mechanism is especially desirable if the faster ways can be accessed in one machine cycle.

Based on the results of the previous section a hit in fast way can switch off the sense amps of the slower ways. We assume this capability as shown in Figure 4. Thus hit latency is determined by the slowest way that produces a hit, plus overhead to route the data after the hit is detected. Based on our simulation parameters and latency data from Cacti 2.0, we assume our asymmetric cache has a 2 cycle latency for the 1024-line and 512-line ways but only 1 cycle for the two 256-line ways.

3.4 Replacement Policy

To achieve smaller average hit latency, it is desirable for most of the cache accesses to hit in the smaller, faster ways of the cache. On the other hand, half of the cached data in our asymmetric cache is stored in the largest, slowest way. As a result, positioning frequently-used data in the smallest cache way is important. Replacement strategies in an asymmetric cache can affect both miss rate and power consumption depending on where heavily accessed data are stored in the cache. One could devise schemes in which the most heavily accessed data are pushed in the smallest ways. However, data movement within the cache would be a significant source of power consumption in this case. In this paper we examine asymmetrical caches with simple LRU replacement policies without data movement.

An LRU replacement strategy is necessary to maintain low miss rates in associative caches, especially when power is a major concern and we want to minimize accesses to the lower parts of the memory hierarchy. LRU replacement is slightly more complex than in conventional caches. We assume here 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 6 shows the LRU scheme of our asymmetric cache. In the asymmetric cache, for the purpose of LRU replacement, we consider that the number of sets is equal to the number of lines in the smallest way. Each set however, comprises 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. In our example design (1024,512,256,256) there are just 256 LRU entries corresponding to the 256 lines of the smallest ways. Each entry contains 4 LRU counters for the largest way (1024 lines), 2 counters for the second largest (512 lines), and 1 counter for each of the smallest ways (256 lines). When new data 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 4 ways specifying a unique path in the mapping tree (see figure 6). 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 was not 4-way but 8-way set associative. In this paper we examine this LRU replacement scheme but we also use a simpler random replacement algorithm (where we replace a random line from the corresponding set). We believe that pseudo-LRU mechanisms are also possible but we have not examined them as of yet.

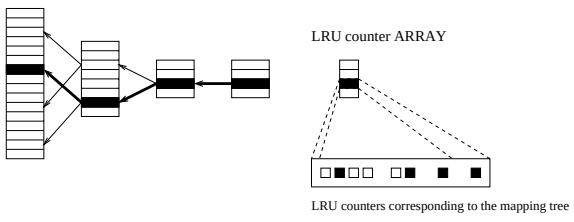

Figure 6: An LRU replacement policy for asymmetric cache

3.5 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 lookup into the sense amplifiers. Thus, the energy consumed on sense amps can be saved. In current caches, sense amps for data array account for a large portion of the total cache power consumption. Figure 7 shows the percentage of cache energy per access attributed to data sense amps for various caches with 32Byte line size in 0.25um technology. The power numbers are from the Cacti 2.0 tool [14]. Across the different cache configurations, about half of the energy per access is consumed on the data sense amps. Thus, early hits on faster ways are much more power-effective than other hits. As shown in Section 4, this significantly reduces the total power consumption of asymmetric caches.

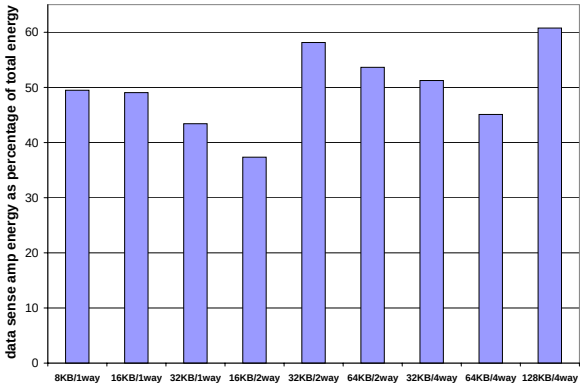

Figure 7: The ratio of data sense amp energy per access for various caches

4 Results

In this section, we will examine simulation results for asymmetric caches and some conventional caches. All caches here have size of 64KB and block size of 32B. The asymmetric cache has 1024, 512, 256 and 256 entries for its 4 ways respectively as described previously. We run the whole SPEC2000 benchmark suite and report their geometric means. Both random and LRU replacement policies are considered here.

4.1 Accesses to Each Way

Figure 8 compares the normalized number of accesses to each way for the asymmetric cache and the 4-way conventional cache.

4.1.1 Random Replacement Policy

With random replacement policy, accesses to each way in a conventional cache are about the same due to their identical size. However, for asymmetric caches, accesses to each way are different. It is interesting to note that the accesses to a way are roughly proportional to the square root of the way size. Particularly, the 256-entry way gets about half as many accesses as the 1024-entry way and the 512-entry way gets about 0.7 times as many accesses. We have observed this behavior for all SPEC2000 benchmarks. This means that smaller ways serve more accesses than what their size would indicate.

4.1.2 LRU Replacement Policy

Using LRU replacement policy, we still observed similar accesses frequencies to each way in conventional caches. For asymmetric caches, accesses are roughly proportional to the size of each way. Specifically, in the asymmetric cache we simulated, the accesses to each way roughly follow a 4:2:1:1 ratio.

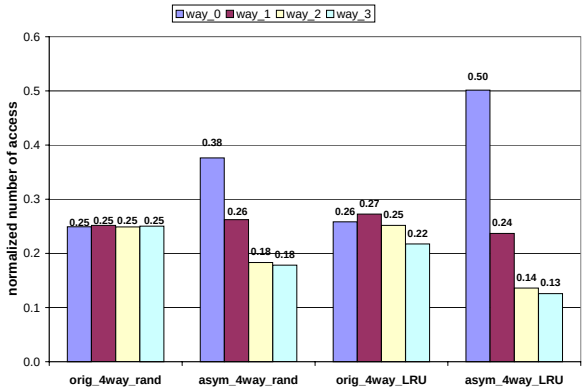

Figure 8: Normalized number of access to each way in asymmetric vs. conventional 4-way caches for SPEC2000

4.2 Miss Rate

Figure 9 compares the miss rate of the asymmetric 4-way set associative cache to conventional 1-way, 2-way and 4-way set associative caches of the same total capacity.

As we expected, the 4-way caches clearly outperform the 1-way and 2-way caches. Within the 4-way caches (conventional and asymmetric), the miss rates are very close but the asymmetric has about a 3% miss rate advantage over the conventional cache for random replacement and a 4% advantage for LRU replacement.

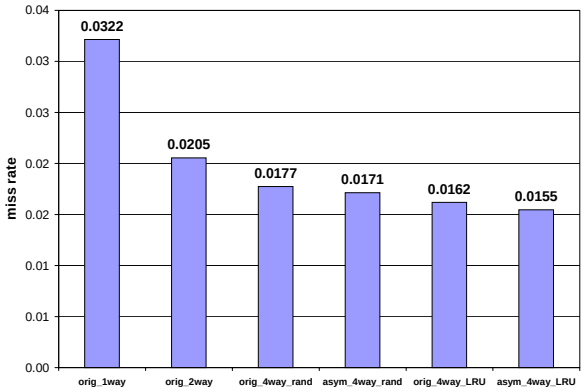

Figure 9: Miss rate of asymmetric vs. conventional 4-way caches for SPEC2000

4.3 Performance

In asymmetric caches, hits on smaller ways have lower latencies than larger ways. However, the impact of this effect on performance is not significant due to two reasons: the aggressive out-of-order execution and the fully pipelined data cache access. In [5], it has been estimated that adding cache latency by 1 cycle only costs about 4% in overall performance. We expect the effect in asymmetric caches is even smaller since accesses hit in the smaller ways only one third of the time. Our simulation results show no noticeable difference for FP benchmarks. For integer benchmarks, we observe a 1% improvement in performance for asymmetric caches compared to conventional caches for both random and LRU replacement policies. Figure 10 depicts the average IPC for SPEC2000 with different 4 way 64KB caches. Finally, we note that in some designs, asymmetric caches may be promising as a way to maintain shorter pipeline depths even as clock cycle times decrease.

4.4 Power Consumption

This section compares the power consumption of our asymmetric caches with the same-sized conventional 4-way cache. Since both caches are 4-way set associative, we assume the energy consumed by the mux drivers and

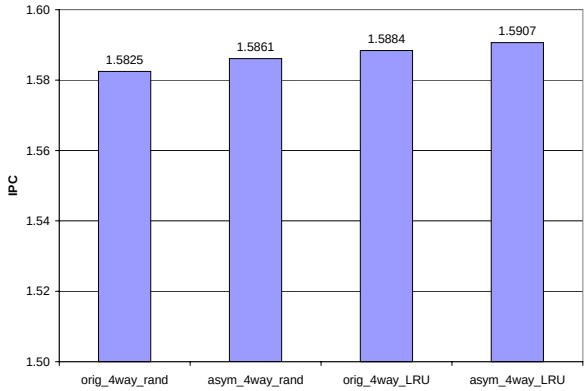

Figure 10: IPC of asymmetric vs. conventional 4-way caches for SPEC2000

the output drivers (see Figure 1 and Figure 4, these drivers account for 1% - 2% energy in a 4-way cache) are similar; we exclude them from our comparison. To calculate access energy for each way, we utilized Cacti 2.0 [14] assuming a 0.25um technology.

	Way 0 1024 entries	Way 1 512 entries	Way 2 256 entries	Way 3 256 entries
Access frequency(%) random replacement	0.3712	0.2588	0.1807	0.1760
Access frequency(%) LRU replacement	0.5015	0.2368	0.1360	0.1257
Access energy (nJ)	3.9799	2.7955	1.9885	1.9885
Data sense amp energy (nJ)	1.4863	1.2132	0.9756	0.9756

Table 3: Access frequency and per-access energy for each way

Table 3 shows the resulting access energy and frequency for each way. 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 amp 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 p(lookup_shorted) * data_sense_amp_energy. Table 4 shows the average access energy of the conventional and asymmetric 4-way 64KB caches. The data clearly illustrate the effectiveness of lookup-shorting. Compared to the conventional cache, an asymmetric cache achieves 17% energy savings with random replacement policy and 13% with LRU replacement policy.

	conventional	asym random	asym LRU
average access energy (nJ)	11.1820	9.3480	9.6939
normalized access energy	1	0.83	0.87

Table 4: Energy of asymmetric and conventional 4-way 64KB caches

5 Conclusions

Many programs can tolerate low associativity for most of their data accesses, but need higher associativity for some small part of their dataset. This is evident in small helper caches such as victim buffers which work well in concert with direct-mapped caches. In this paper we take a different approach to exploit the same phenomenon: we modify set-associative caches so that different ways have different sizes. We call the resulting architectures asymmetric set-associative caches. In 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.

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 lookup at the slower cache ways.

Thus, asymmetric caches have the benefit of lower power consumption in the smaller ways while maintaining the same miss rate as conventional caches. In our experiments we actually see a 3% reduction in the SPEC2000 average miss rate. By immediately terminate lookups in larger ways when detecting a hit on smaller ways, the average cache access energy is reduced by 17% 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] A. Agarwal and S. Pudar. Column-associative caches: A technique for reducing the miss rate of direct-mapped caches. In *Proc. ISCA-20*, 1992.

[2] F. Bodin and A. Seznec. Skewed associativity enhances performance predictability. In *Proc. ISCA-22*, June 1995.

[3] D. Burger, T. M. Austin, and S. Bennett. Evaluating future microprocessors: the SimpleScalar tool set. Tech. Report TR-1308, Univ. of Wisconsin-Madison Computer Sciences Dept., July 1996.

[4] P. W. Diodato. Personal communication, 2001.

[5] L. Gwennap. Digital 21264 sets new standard. *Microprocessor Report*, pages 11–16, Oct. 28, 1996.

[6] M. G. Johnson Kin and W. H. Mangione-Smith. The filter cache: An energy efficient memory structure. In *Proc. Micro-30*, Nov. 1997.

[7] N. Jouppi. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. In *Proc. ISCA-17*, May 1990.

[8] T. Juan, T. Lang, and J. Navarro. The difference-bit cache. In *Proc. of the 23rd Int'l Symp. on Computer Architecture*, June 1996.

[9] R. E. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, 1999.

[10] Luis Villa, Michael Zhang and Krste Asanovic. Dynamic Zero Compression for Cache Energy Reduction. In *Proc. Micro-33*, Dec. 2000.

[11] V. Milutinovic et al. The Split Temporal/Spacial Cache: Initial Performance Analysis. In *SCIzzL-5*, 1996.

[12] J. Peir, Y. Lee, and W. Hsu. Capturing Dynamic Memory Reference Behavior with Adaptive Cache Topology. In *Proc. ASPLOS-VIII*, Nov. 1998.

[13] M. Prvulovic et al. The Split Spatial/Non-Spacial Cache: A performance and Complexity Analysis. In *IEEE TCCA Newsletter*, 1999.

[14] G. Reinman and N. Jouppi. Extensions to cacti. Unpublished document, 1999.

[15] J. Sahuquillo and A. Pont. Splitting the Data Cache: A Survey. In *IEEE Concurrency*, 2000.

[16] S. Sair and M. Charney. Memory behavior of the SPEC2000 benchmark suite. Technical report, IBM, 2000.

[17] A. Seznec. DASC cache. In *Proc. HPCA-1*, Jan. 1995.

[18] The Standard Performance Evaluation Corporation. WWW Site. http://www.spec.org, Dec. 2000.

[19] S. Wilton and N. Jouppi. An Enhanced Access and Cycle Time Model for On-chip Caches. In *WRL Research Report 93/5, DEC Western Research Laboratory*, 1994.