Low-Energy Data Cache Using Sign Compression and Cache Line Bisection

Nam Sung Kim, Todd Austin, Trevor Mudge

{kimns, taustin, tnm}@eecs.umich.edu
Advanced Computer Architecture Lab,
The University of Michigan,
1301 Beal Ave. Ann Arbor, MI, 48109-2122

Abstract

Cache accesses consume a significant portion of total energy dissipation in modern microprocessors. In this paper, we introduce a new technique for data cache energy reduction, which exploits the prevalence of small values and the inefficiency of using a full word for their storage. Simulation results show that we can reduce total data cache energy by around 23% for the SPEC benchmarks without significant performance degradation. We also propose a modified data compression cache architecture utilizing empty cache space that is not used after data compression. This proposed architecture improves the cache miss rate, which results in improved performance and reduced energy dissipation when accessing the next level of the memory hierarchy. We show that this scheme has an energy saving of about 19%.

1. Introduction

Energy dissipation is an increasingly important design constraint in a wide range of processors, from those intended for mobile use, all the way up to high-performance processors for high-end servers [1]. Within a typical processor, cache accesses consume a significant fraction (30%-60% [2, 3]) of the total energy dissipation. As new generations of processors incorporate ever larger caches this percentage will continue to grow.

A large portion of cache energy is dissipated in driving the bit-lines, which are heavily loaded with storage cells, and so most cache energy reduction techniques have concentrated on reducing bit-line energy dissipation. One approach is to reduce the bit-line capacitance switched on each access by using a combination of sub-banking, segmented bit-lines, and hierarchical bit-lines [4]. A complementary approach is to limit the voltage swing on the bit-lines during read accesses by pulsing word-line drivers [5, 6]. According to [7, 8, 14], over 70% of the bits that are read from or written to the data cache are zero, furthermore 75% of the values are small, requiring only 16 or 8 bits of significance. The rest of the upper 16 bits or 24 bits can be represented with just a single sign bit. We will refer to this as sign compression even though it may be applied to data that is not numerical but happens to have upper bytes that are all zeros or all ones.

In this paper we propose a new technique that makes use of sign compression to reduce energy dissipation in caches. It is particularly suited to data caches where data has to be both read and written, and thus must accommodate changes in the number of significant digits. Read-only instruction caches and tables of fixed data can be handled by other means that allow greater compression [9].

Our approach stores the words in some cache lines in sign-compressed form. Specifically, we compress those words whose upper half is either all zeros or all ones into a half-word and sign bit. In these cases the full capacity of each word is not used, and, as a result, we can reduce the energy dissipation caused by bit-line discharging and leakage energy by turning on only the lower half of each word of a cache line when we access the line. We show that a significant fraction of the cache lines can be compressed in this way so that they use only about half of a normal line. In the less frequent case that the cache line cannot be compressed, it is stored in its uncompressed form. When a line is accessed, energy is saved by accessing only the lower half of each word, the assumption being that it is compressed. If this is not the case, the rest of the cache line is accessed in the next cycle. We show that this penalty is quite small.

The scheme proposed in [7] reduces data cache energy dissipation by compressing bytes that are all zeros into a single bit and by accessing only the remaining bytes. However, they apply the compression scheme for every byte of the cache line, which increases the complexity of the cache architecture for only a modest reduction in energy. They also lose the opportunity to compress bytes of all-ones. In [8], they take advantage of bytes containing all ones and all zeros, however, their cache access mechanism causes a notable amount of performance degradation because they access only one byte in one cache access cycle. Furthermore, both of the compression schemes leave the empty space caused by the data compression unused.

Although the technique proposed in [14] uses a similar compression scheme and utilizes the unused cache space by the compression, they overlooked the fact that the compressibility can be increased significantly by adding some extra storage space and allowing slack for the incompressible upper half-words in the fetched cache lines. In additional experiments we demonstrate that this space can be gainfully employed to reduce energy.

The rest of this paper is organized as follows. Section 2 presents the basic idea showing how sign compression can be applied to compressing cache lines. Section 3 elaborates on the basic idea, showing how it can be built into a low energy cache architecture using cache line bisection. Section 4 presents an alternative architecture that stores additional data in the unused cache space remaining after compression. A reduced miss rate results that reduces energy dissipation by decreasing the number of accesses to the next level of the memory hierarchy (the L2 cache in our experiments). Section 5 describes the simulation environment, the implementation, and presents experimental data showing energy reduction and the performance impact of the proposed cache architectures. Section 6 concludes the paper and suggests future work.

2. Sign-Compression for Data Caches

Figure 1 shows the basic idea behind our data compression scheme for the data fetched on a cache miss. The lower half of each word fetched during a cache line miss is not changed, but the upper half-words are replaced by a zero or one when the upper half-words are all zeros or all ones respectively. When, all the upper half-words are compressible in this way, we can represent the fetched cache line with only half the number of bits normally reserved for the cache line plus a few sign extension flag bits. Fortunately, it is often the case that all the upper half-words in the fetched line are compressible. However, our preliminary experiments to examine the compressibility of cache lines reveal that a significant number of additional cache lines can be compressed if we allow some tolerance in our compression scheme. Specifically,

FIGURE 1. **Sign compression for a cache line. The words in this diagram are 16 bits, and up to one uncompressed word is tolerated per line.**

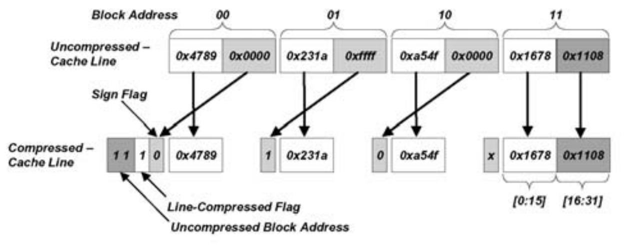
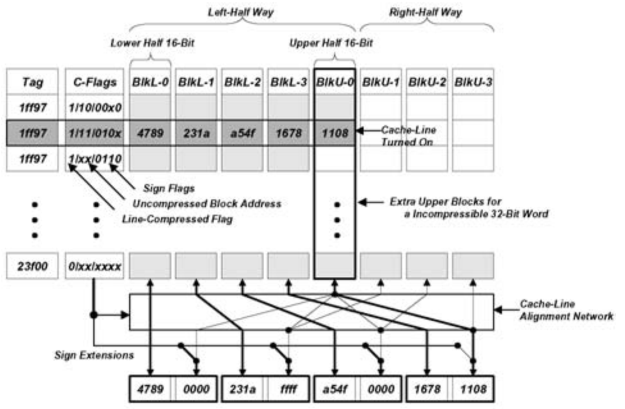


FIGURE 2. **Line compression with sign compression and cache line bisection. The line has four words of 16 bits, and a tolerance of up to one uncompressed word in compressed lines.**



we allow a few incompressible upper half-words to be included by using extra memory space to store the incompressible upper half-words. To support this scheme we need additional flag bits to indicate the position of the uncompressed word(s) in the line, a flag bit to represent whether or not the line contains any compressed words, and a sign bit for each compressed word. The number of uncompressed words allowed in a cache lines is a design parameter—Figure 1 shows a line that tolerates up to one uncompressed word.

3. An Architecture for Low-Energy Caches

3.1 Line compression using sign-compression and cache line bisection

Figure 2 shows a low-energy data cache architecture using cache line bisection to store sign-compressed words. To reduce energy dissipation from bit-line discharge for those upper half-words whose information can be compressed into a sign bit, we divide the cache line into two parts (line bisection) and access only one part at a time. The first to be accessed is the left-half way containing the tags for the cache line, flags for the compression information, the lower half-words of the cache line, and a part of the upper half-words used to store the pre-defined number of incompressible upper half-words for compressed lines.

Three types of cache line result: 1) a line which is uncompressed; 2) a line in which every word is compressed; and 3) a line in which one or more words are not compressed. Figure 2 shows a cache line of four words of 32 bits and half-words of 16 bits with just one word allowed to be uncompressed. As noted before, the number of uncompressed words tolerated in compressed cache lines is a design parameter. In Figure 2, 5/8 of the data cells in the cache array are accessed when the left-half way is first accessed.

The design requires some special circuitry for the data compression and decompression between the cache and the processor. First, we need a circuit for checking the compressibility of the fetched line on a cache miss. This can be quite simple—it need only detect all ones (AND) or all zeros (NOR) for every upper half-word of the fetched line. The more critical circuitry is the data aligner needed to route the compressed lines into the cache and to decompress cache lines when they are read into the processor. Figure 2 provides a sketch of how this might work. It can be implemented by multiplexers controlled by the compression flag bits. They are similar to those that would be required for a set-associative cache, but their control is a little more complex. In fact, the way multiplexers can serve double duty if a set-associative cache is implemented.

The access timing of the proposed cache architecture is shown in Figure 3. In the first cycle of the cache access, the cache decoder activates the indexed cache word-line and the left-half way is retrieved including the cache line tag and compression flags. As soon as it can be determined that the accessed cache line is not compressed, the right-half way is pre-charged allowing the upper half-words to be read out in the following cycle. This timing arrangement increases the cache access latency by one cycle whenever we access a cache line that is not compressed. We model this penalty in our simulations and show that it has only a small negative effect on performance.

We noted that a significant portion of the energy consumed by the cache is dissipated by the bit-lines, which we can reduce to nearly half with our proposed cache architecture. This suggests we should increase the number of compressible lines by allowing more incompressible upper half-words in the cache line to reduce the number of right-half way cache accesses and thus reduce the energy dissipated in the bit-lines. However, allowing too many extra incompressible upper half-words has a negative effect at some point, because the energy savings are offset by the energy needed for the extra incompressible upper half-words in the left-half way. Data cache lines are written as well as read so it can happen that a compressed line can change to an incompressible one as a result of a write. Conversely, lines that are not compressed can become candidates for compression as writes occur. Fortunately, we show that cache writes do not result in a noticeable decrease (or increase) in the number of compressed lines.

3.2 Compression experiments

Figure 4 shows our experimental results using SPEC benchmarks to measure the percentage of accessed cache lines that can be compressed for different tolerances. The experimental cache configuration was a 16-KB, 4-way, set associative cache with a 32-byte line. Among the cache configuration parameters, the cache line size affects the compressibility of the fetched cache line, because there are more chances that all the words will have non-significant upper

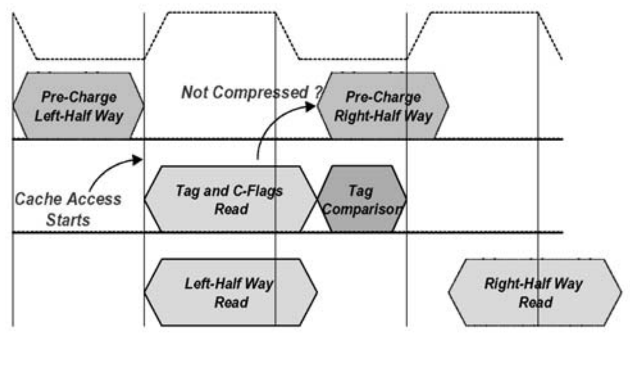


FIGURE 3. **Compression cache management and access timing**

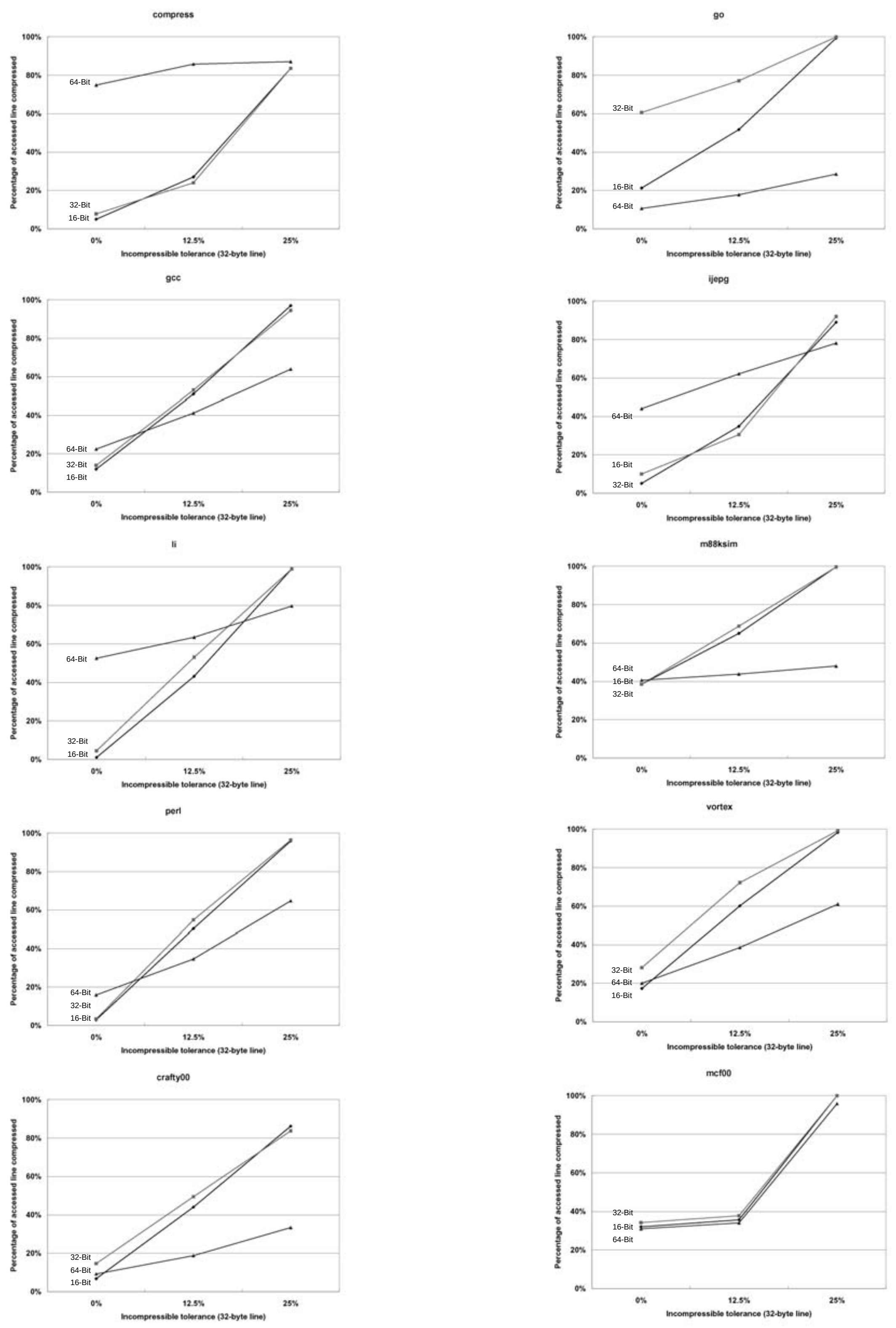


FIGURE 4. Compressed cache line access rates: The percentage of compressed cache lines that are accessed for three tolerances and three words sizes. Ten SPEC benchmarks are shown.

bytes, the smaller the cache line. To avoid this extreme we have chosen a fairly large common line size, although as we will show smaller lines can provide greater compression. The experiments also examined the effect of different words sizes on compressibility. We considered 16-bit, 32-bit and 64-bit words. The word size is simply the subdivision of the cache line which can be compressed. Thus in the case of a 16-bit word each upper byte per 16 bits can be recoded as one or zero, and for a 32-bit word the two upper bytes can be recoded as one or zero, etc. In addition, we chose three tolerance schemes 0%, 12.5% and 25% for the number of the allowable incompressible words. The tolerance indicates the fraction of a line that can be stored in the upper half-words of the left-half way of the cache. For example, a 12.5% tolerance for a 32-byte cache line allows two upper 16-bit words (4-bytes) to be stored in the left-half way of the cache.

Tolerance	0 bytes	4 bytes	8 bytes
16-bit word	14.18%	46.28%	94.78%
32-bit word	21.56%	52.06%	95.79%
64-bit word	32.11%	44.00%	64.02%

TABLE 1. Average ratio of compressible cache lines to uncompressed ones for cache accesses.

As expected we can obtain more compressible cache lines as we allow more incompressible upper half-words in the lines fetched on a cache miss, and we can confirm this fact from the experimental results shown in Figure 4. Furthermore, a 64-bit word unit works well when no tolerance is allowed, because only four consecutive words have to be compressible. When a smaller size is used for the compressible word size, there are more chances to obtain individual compressible words. However, this has to be balanced against the decreasing likelihood of more consecutive words having upper halves that are all-zeros or all-ones. Our experiments show that shorter words compress better as we allow a higher tolerance for incompressible words. However, the 32-bit word size outperforms the 16-bit word size, in most cases, because the base-line architecture is a 32-bit machine and most of the significant values need more than 8 bits. This can be seen clearly in Table 1 which summarizes the results of Figure 4.

Tolerance	0 bytes	4 bytes	8 bytes
16-bit word	22.62%	64.36%	95.16%
32-bit word	47.94%	68.40%	93.86%
64-bit word	57.18%	65.88%	74.14%

TABLE 2. Average compression ratios for cache lines fetched on a miss.

The compressibility of cache miss fetches is also important. The percentage of compressible miss fetches is shown in Table 2. The statistics show that the compressibility of cache lines fetched at misses is slightly higher than those shown previously for cache accesses, suggesting that some of the initially compressed cache lines become uncompressed during program execution as a result of store operations. In this paper, we only consider the benefits of compression of the top level cache structures, however, these results suggest that compression could be gainfully applied to the memory system busses and deeper levels of the memory hierarchy. If the data is sign-compressed at the next level of the memory, it can be brought over the memory-cache bus in compressed form. This will save bus energy, especially if the bus is off-chip. In addition, compressing the L2 cache structures will likely result in benefits similar to those described in this study.Before we leave this technique, we examine the effect of varying the cache line size. Figure 5 shows the average compression ratios for cache lines fetched on a cache miss for 8, 16, and 32-byte cache line block sizes. In this experiment, we used 16 KB caches and did not assign any extra blocks for the incompressible words in the fetched cache line—zero tolerance for the incompressible words. Therefore, we cannot compress the fetched line if there exists any incompressible words among the words in the fetched line.

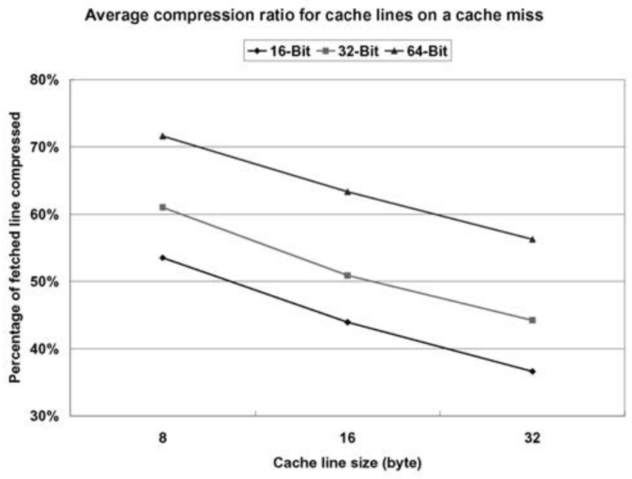


FIGURE 5. Average compression ratios for cache lines fetched on a cache miss for 8, 16, and 32-byte cache line block sizes: A 16-KB cache was used with ten SPEC benchmarks.

It can be seen that the compressibility of the fetched cache lines decreases as the number of the bytes in the cache line increases, because the probability of obtaining contiguous compressible words decreases. In addition, a 64-bit word size shows the best compressibility, because it has a smaller number of words than other word sizes for the same cache line block size. However, this is only true for zero tolerance. We can obtain a different result if we allow some tolerance for the incompressible words as we saw in the experiments of Table 2 in compressing the fetched line since then we lose chances to compress the line with finer granularity if we use the 64-bit compression word size.

The point to notice for the work presented in this paper is that we chose a 32-byte cache line to be representative of some existing systems rather than use small line sizes to inflate the results.

4. A Cache Architecture that Uses the Empty Right-half Way

4.1 Using the empty right-half way

Although we can save energy by preventing the discharge of the bit-lines in the right-half way for compressed cache line accesses, like [7, 8] we waste the cache memory space which is not utilized by the compression scheme. From the experimental results shown in Section 3 it was seen that we can represent many cache lines with only half of the cache line space, if we allow some tolerance for the incompressible words in each cache line. This suggests that we can increase the effective cache size, and accordingly performance, if we provide an independent tag array for the right-half way. Figure 6 shows a modified data compression cache archi-

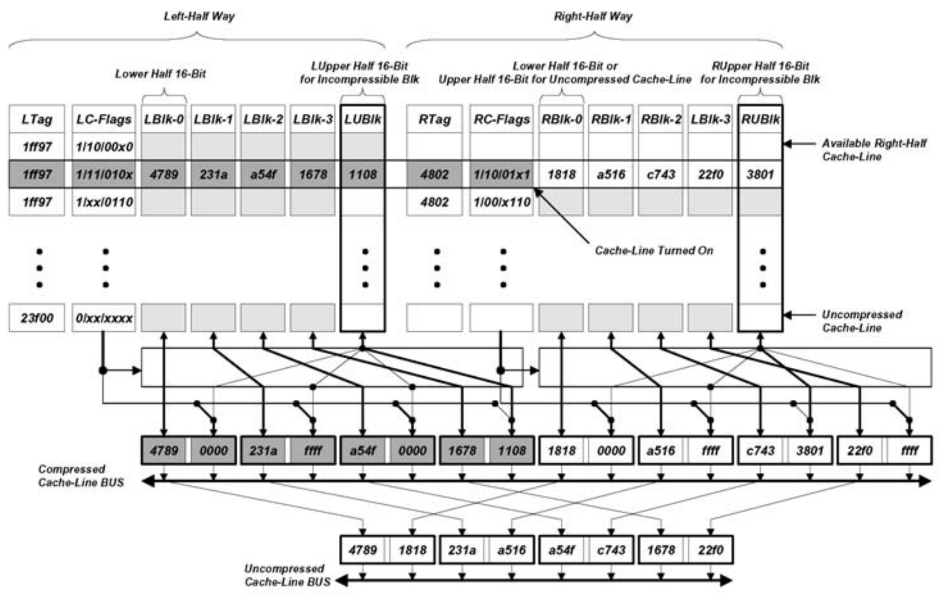


FIGURE 6. **A modified cache architecture that uses the empty right-half way.**

tecture based on this idea. Each half way has its own tags allowing compressed cache lines to be stored in both half ways. The effective cache size is thus increased at the cost of an additional tag and compression flags for the right-half way, plus incompressible upper half-word storage for both ways of the cache line. Using both ways and adding a tag array increases energy consumption; however, the fraction of the tag bit-lines in tag arrays are relatively small compared to the number of the bit-lines in data arrays, and the effective increase of the cache size reduces the number of accesses to the next level in the memory hierarchy—an L2 cache in our experiments. It is this reduction that saves energy by decreasing the activity in the L2 cache.

4.2 Cache management

When accessing this modified cache, the tag bit-lines of both half ways are examined to check the availability of the requested cache line. To keep energy consumption to a minimum, only the bit-lines of one of the halves are pre-charged based on a most recently used (MRU) mechanism that exploits the temporal locality of cache access. A performance and energy penalty occurs when there is an MRU miss or when the accessed cache line is not compressed.

Whenever a cache miss occurs, the L2 cache is accessed and the requested cache line is fetched. During the L2 access the availability of empty right-half ways are checked among the indexed sets before we select a cache line for the replacement. If there exists an available empty right-half way and the fetched cache line is compressible we store the compressed line in the available right-half way without replacing the existing cache line. If there does not exist an available empty right-half way or the fetched cache line is not compressible, we proceed with a normal cache replacement sequences as shown in Figure 7.

Occasionally store operations can make a compressed line incompressible. In such cases, it is necessary to evict the one of the compressed line if the compressed lines occupy both cache sub-lines. During the eviction process the evicted line must be written back if the status of the line is dirty. If both ways are dirty this requires additional write-back cycles. However, this case is rare according to the experimental results that will be shown in Section 5.

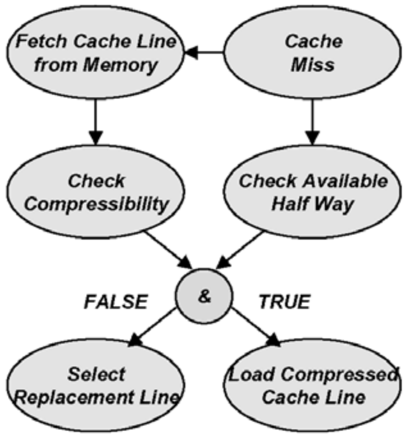


FIGURE 7. **Cache miss handling for the modified cache architecture.**

4.3 Experimental results when the empty half line is used

There are three types of fetched lines. One is an incompressible line that occupies both half ways. Another is a compressible line that occupies the left-half way of the cache line. The other is a compressible line that occupies the right-half way of the cache line. The summation of the ratios of both compressible categories represents the compressible ratio for lines fetched on a cache miss. Figure 8 shows the distributions of the fetched cache line types. The word size is 32-bit and 25% additional cache space is available for the incompressible tolerance of each half way.

The results in Figure 8 show that a significant fraction of the fetch lines fit into the right-half way of the cache line, effectively improving cache capacity.

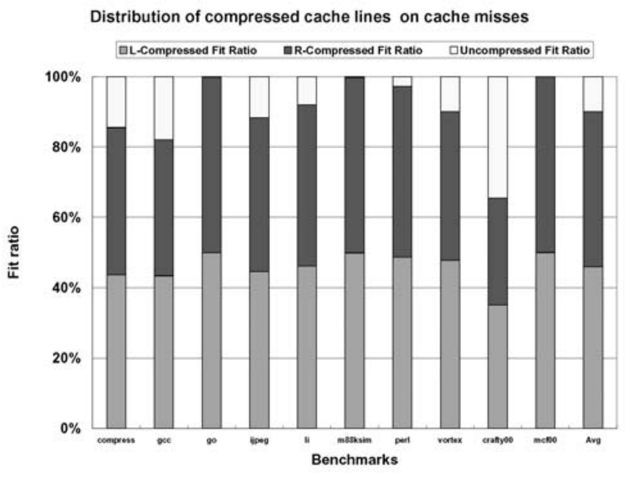


FIGURE 8. Distribution of ratios for cache lines fetched on a cache miss.

In order to discover how much the cache miss rate improves, the miss rates were measured with 4KB, 8KB, 16KB, and 32KB caches size and 1-, 2-, and 4-way associativities while running the SPEC benchmarks. The experimental results are shown in Figures 9 and 10. For comparison the miss rates are also shown for a normal cache of the same size and a normal cache of double the size which has twice as much associativity as the normal cache. The results in the figures show that, in most cases, the miss rate of the modified data compression cache is closer to the miss rate of the double-sized cache than the same-sized cache.

5. Evaluation of the Low Energy Cache Architectures

5.1 Evaluation methodology

The evaluation methodology combines detailed processor simulation for performance analysis and for gathering event counts, and analytical modeling for estimating the energy dissipation for both the conventional caches and those employing the compression strategies. We used the SimpleScalar toolset [11] to model an out-of-order speculative processor with a two-level cache hierarchy. The simulation parameters, listed in Table 3, roughly correspond to those of a present-day high-end microprocessor such as the HP PA-8000 or Alpha 21264.SimpleScalar does not move actual data between memory levels, but only captures the behavior of the address streams. We need actual data values to validate the energy reduction and performance degradation of the proposed data compression cache architectures, because we have to be able to calculate the compressibility of the lines. Thus SimpleScalar was modified to trace the actual data streams between the memory levels. Furthermore, a configurable cache miss handler, data compressor, and decompressor were added for the proposed data cache compression architectures.

In addition, we required a cache energy estimator to measure the energy usage of the conventional cache as well as the proposed cache architecture accurately. The energy usage of the L1 data cache and unified L2 cache was computed with Wattch [12]. This model in turn calculates cache energy dissipation using event counts from the SimpleScalar simulations and technology and layout parameters from CACTI, which has been previously calibrated against HSPICE [13]. We use simple event counts because we are comparing energy rather than power dissipation. The SimpleScalar execution-driven simulations employ ten SPEC benchmarks from the SPEC95 and SPEC2000 benchmark suites: *compress*, *gcc*, *go*, *ijpeg*, *li*, *mk88sim*, *pearl*, *vortex*, *crafty*, and *mcf*. We run each benchmark for 1 billion instructions. Only L1 data cache and the unified L2 cache energy dissipations are considered, because the L1 instruction cache and the main memory energy dissipation do not change significantly with the proposed compression architecture.

Parameter	Value
fetch width	4 instructions
fetch queue	4 instructions
fetch speed	1x
decode width	4 instructions
issue width	4 instructions
commit width	4 instructions
branch prediction	bimodal, 2K
BTB	512 entry, 4-way
RAS	8 entry
RUU size	16 entry
LSQ size	8 entry
LSQ size	8 entry
integer ALUs	4
integer mult/divs	1
floating point ALUs	1
floating point multi/divs	1
L1 instruction cache	16KB, 2-way set associative, 32B line block, LRU, 1 cycle latency, write-back
L2 unified cache	256KB, 4-way set associative, 64B line block, LRU, 6 cycle latency

TABLE 3. Simulation parameters.

Figure 11-(a) shows compressible cache line fetch ratios for the SPEC benchmarks from Figure 4 for a 32-bit word size. This is for the first cache line bisection scheme. The benchmarks go, m88ksim, and mcf show a very high compressibility for different tolerances, because they are control-intensive programs and most of data used in these programs are small values. Furthermore, the data in the cache line for these three programs are very sparse, because there is little data used in control-intensive programs. For the 25% tolerance scheme, more than 85% of the fetch lines can be compressed over all the benchmarks, which outperforms the 12.5% scheme by nearly a factor of two. (This promises good results in the data compression architecture where the empty right-half way is used, too.)

As noted earlier, we must check how many cache accesses, rather than fetches, are to compressed lines, because this metric is directly related to the energy dissipation and performance degradation. Figure 11-(b) shows that the 25% tolerance scheme again significantly outperforms the 12.5% one. Therefore, although this scheme uses 12.5% more resources, we will use it for the compression cache architecture of Section 4.1 that uses the empty right-half way, when we compare the energy reduction and performance with the conventional and double-sized caches. The total extra cache memory space increases by 50% if we use the 25% tolerance scheme for each half way. This size increase places the cache midway between the normal and double-sized caches, making comparisons easier.

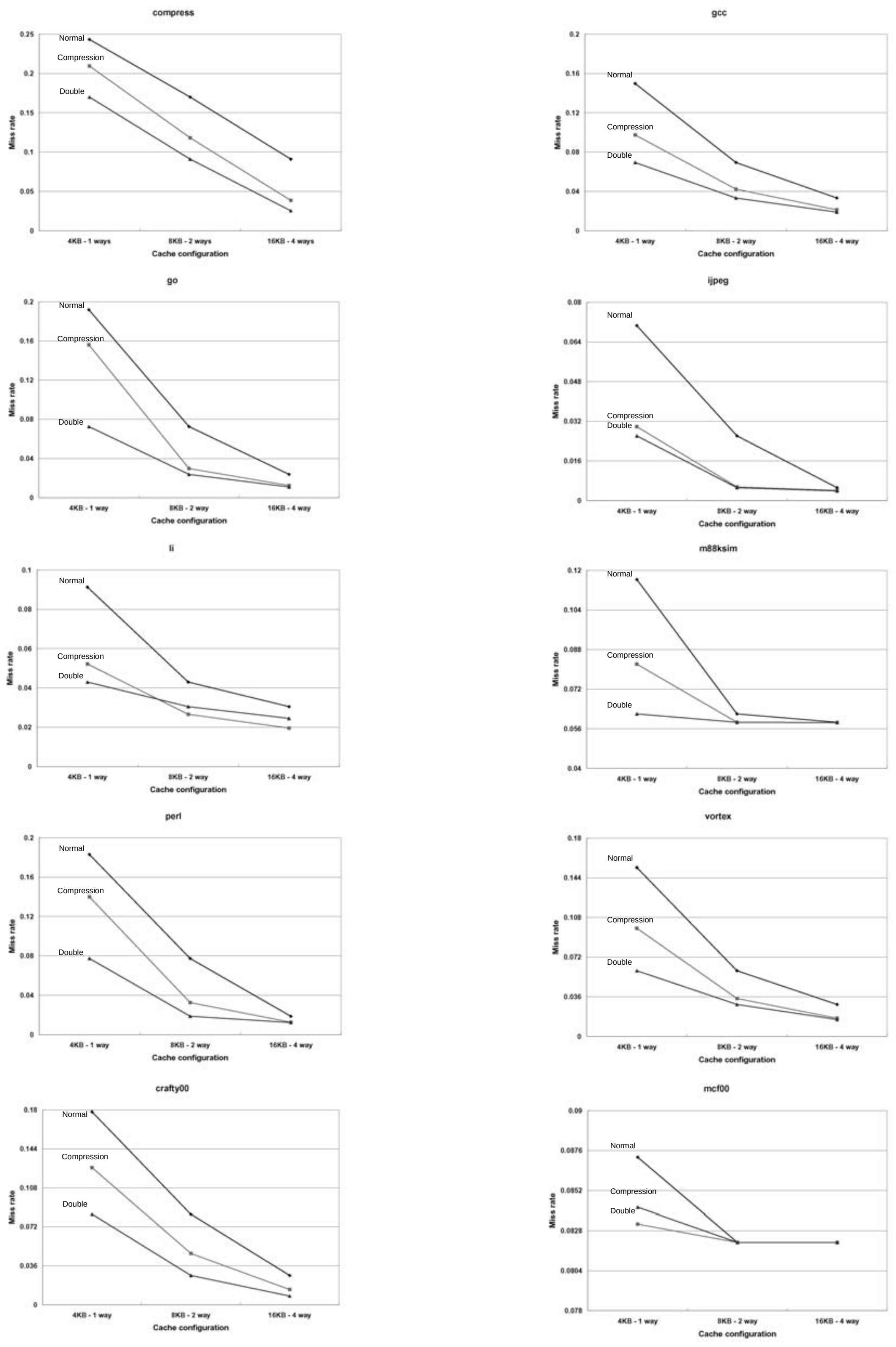


FIGURE 9. **Cache miss rates improvement for 4KB direct, 8KB 2-way, and 16KB 4-way configurations. Normal caches of the same and double size are shown for comparison.**

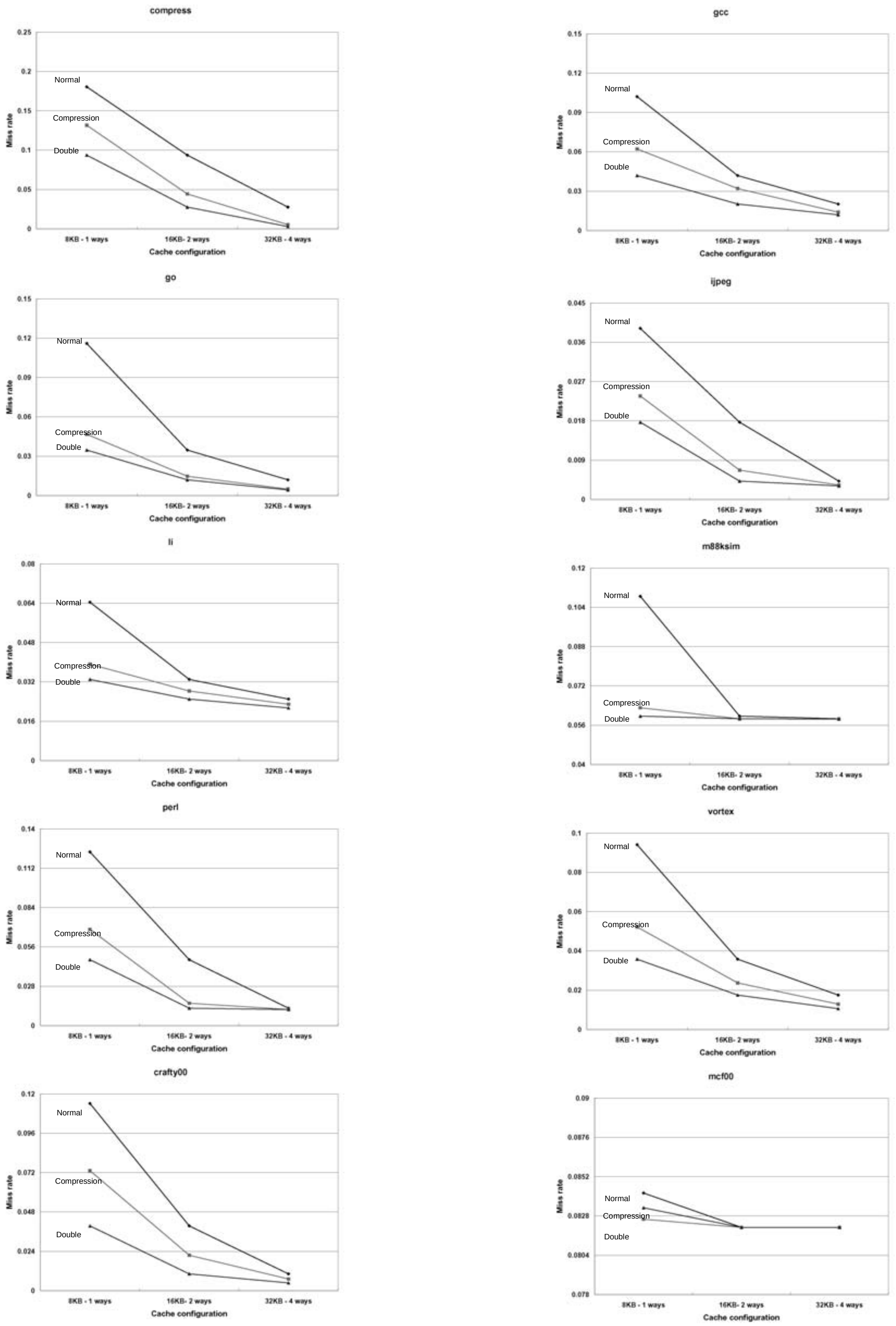
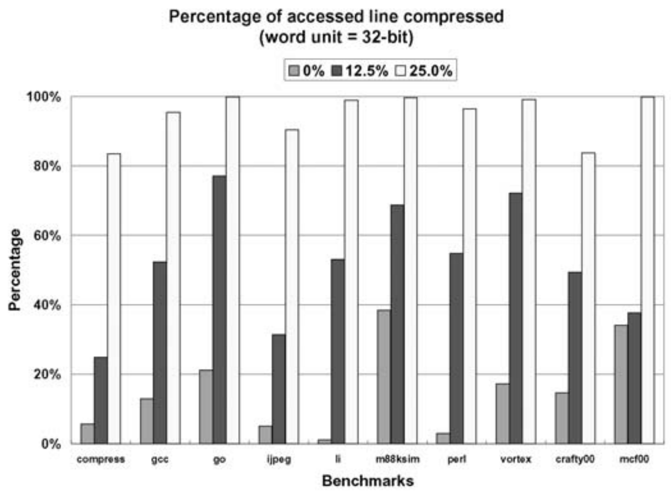
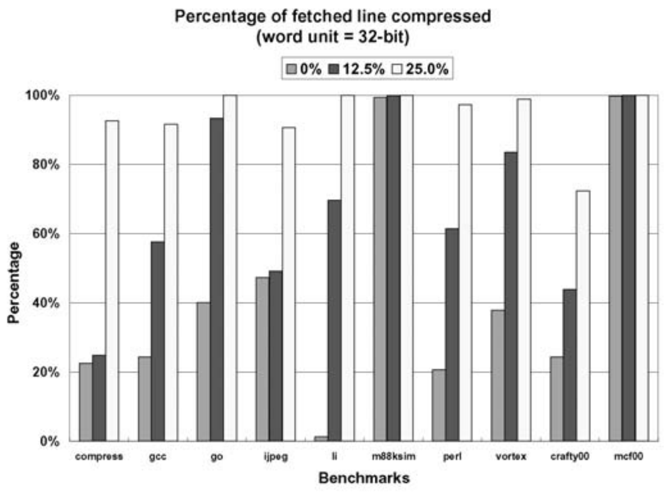


FIGURE 10. Cache miss rates improvement for 8KB direct, 16KB 2-way, and 32KB 4-way configurations. Normal caches of the same and double size are shown for comparison.

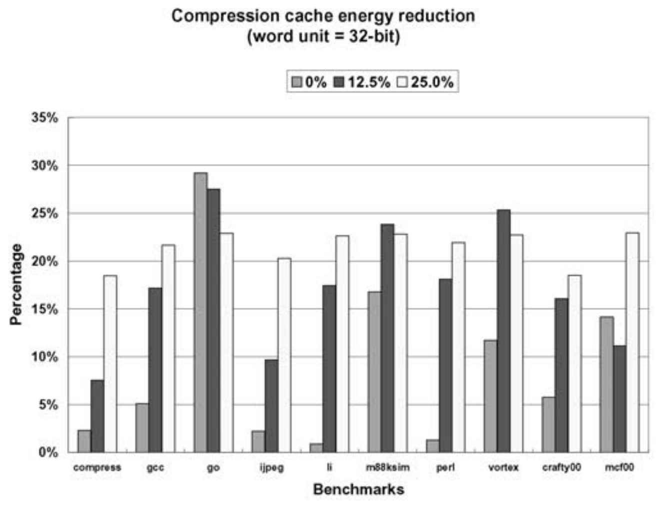


(a)

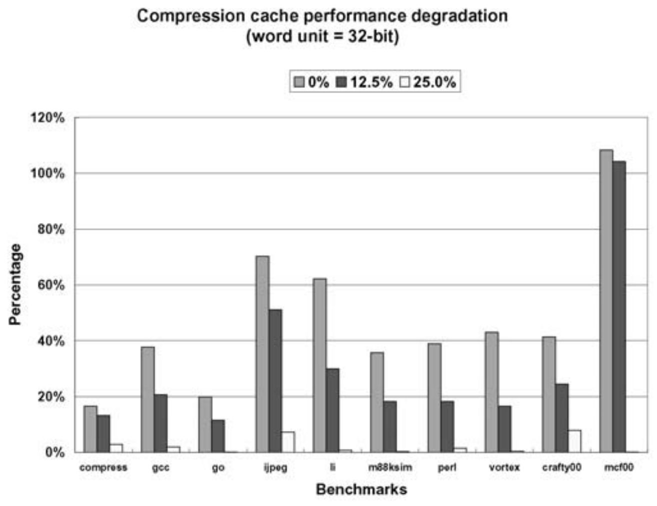


(b)

FIGURE 11. Percentage of compressible cache lines fetched (a) and compressible cache lines accessed (b) for the SPEC benchmarks. Three tolerances are shown.



(a)



(b)

FIGURE 12. Energy reduction (a) and performance degradation (b) of the cache line bisection architecture.

5.2 Energy and performance impact

Simulation results in Figure 12 show the energy reduction ratio and performance degradation of the cache line bisection architecture of Section 3.1 compared to a conventional cache architecture having a 16KB, 4-way cache with a 32-byte line size. In this architecture we do not need extra memory space except for the compression tags which are very small compared to the entire data and tag array. They introduced about 5% overhead when a 32-bit word size is used. The average energy reduction ratio over the SPEC benchmarks is 23% for the 25% tolerance scheme. Furthermore, there is only 1.7% performance degradation in the 25% tolerance scheme because most of the cache accesses are compressed and only a few penalty cycles are incurred.

Figure 13 shows the energy reduction ratio for the architecture of Section 4.1 that attempts to use the empty right-half way. The cache is 16KB, 4-way with a 32-byte line size and a 25% tolerance scheme in each half way. Recall that this does not include the energy dissipation of driving the bus between the L1 and L2 caches; it just measures the savings from reducing the number of accesses to L2. For a larger L2, the savings would be larger.

6. Conclusion and Future Work

This paper presents two cache compression architectures to reduce cache energy. They take advantage of the high occurrence of all-ones and all-zeros that primarily occur as a result of the large number of small values typically stored in an L1 data cache. The first technique uses a small amount of additional hardware embedded in the cache RAM array to manage the reading and writing of compressed values. Simulation results show a 23% energy reduction on data cache accesses with a 1.7% performance degradation and a 5% increase in area.

In the second technique, we use about 50% more RAM array to provide overflow bytes so that we can utilize the unused portion of the cache from the first scheme. This technique improves the cache miss rate significantly compared to a conventional cache (although of course it uses more memory) and approaches the miss rate of a double-sized conventional cache, for small caches. The impact of reducing the miss rate is directly proportional to the energy dissipation of accessing the next level of the memory hierarchy (L2 in our case). Simulation results show an average 19% energy reduction when we account for fewer misses accessing the L2 unified cache. The performance degradation is insignificant or shows a slightly better performance than the conventional cache with a 58% area overhead.

Although there is significant energy dissipation from driving the bus capacitance between the L1 and L2 caches whenever the L2 cache memory is accessed, we did not consider it. This effect will also become more significant as the process technology shrinks. Thus our savings are notably conservative and including the bus energy loss is work that remains for the future.

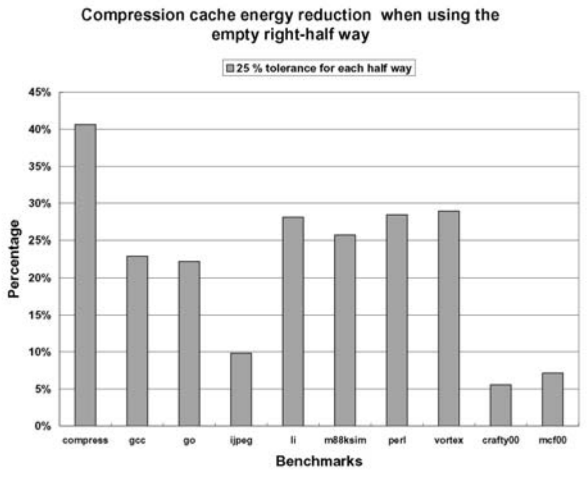


FIGURE 13. Energy reduction ratio when using the empty right-half way.

7. Acknowledgements

This research is supported in part by DARPA under contract AFRL F33615-00-C-1678, SRC under contract SRC-2001-HJ-904, and National Science Foundation CADRE program under contract EIA-9975286.

References

[1] T. Mudge. Power: A first class design constraint. *Computer*, vol. 34, no. 4, April 2001.

[2] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessor. *IEEE Journal of Solid State Circuits*, Sep. 1996.

[3] N. Vijaykrishnan, M. Kandemir, M. Irwin, H. Kim, and W. Ye. Energy-driven integrated hardware-software optimization using SimplePower. *Proc. 27th Int. Symp. on Computer Architecture*, June, 2000.

[4] K. Ghose and M. B. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffer and bit-line segmentation. *Proc. Int. Symp. on Lower Power Electronic Devices,* Aug. 1999.

[5] B. Amrutur and M. Horowitz. Techniques to reduce power in fast wide memories. *Proc. Int. Symp. on Lower Power Electronic Devices,* Oct. 1994.

[6] S. Santhanam et al. A low cost 300-MHz RISC CPU with attached media processor. *IEEE Jour. of Solid-State Circuits,* Nov. 1998.

[7] L. Villa, M. Zhang and K. Asanovic. Dynamic zero compression for cache energy reduction. *Proc. 33th Int. Symp. on Microarchitecture*, Dec. 2000.

[8] R. Canal, A. Gonzalez, and J. Smith. Very low power pipelines using significance compression. *Proc. Int. Symp. on Microarchitecture,* Dec. 2000.

[9] C. Lefurgy, E. Piccininni, T. Mudge. Reducing code size with run-time decompression. *Proc. of the 6th Int. Symp. on High-Performance Computer Architecture*, Jan. 2000.

[10] C. Benveniste, P. Franaszek, and J. Robinson. Cache-memory interfaces in compressed memory systems. *Proc. 27th Int. Symp. on Computer Architecture,* June 2000.

[11] D. Burger and T. Austin. *The SimpleScalar Toolset, Version 2.0.* Tech. Rept. TR-97-1342, Univ. of Wisconsin-Madison, June 1997.

[12] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimization. *Proc. 27th Int. Symp. on Computer Architecture*, June 2000.

[13] S. Wilton and N. Jouppi. An Enhanced Access and Cycle Time Model for On-Chip Caches. Tech. Rept. 93/5, Digital Western Research Lab., July 1994.

[14] J. Yang, Y. Zhang, and R. Gupta. Frequent Value Compression in Data Caches. *Proc. 33th Int. Symp. on Microarchitecture,* Dec. 2000.