Restrictive Compression Techniques to Increase Level 1 Cache Capacity

Prateek Pujara
Dept of Electrical and Computer Engineering
Binghamton University
Binghamton, NY 13902
ppujara1@binghamton.edu

Aneesh Aggarwal
Dept of Electrical and Computer Engineering
Binghamton University
Binghamton, NY 13902
aneesh@binghamton.edu

Abstract

Increasing cache latencies limit L1 cache sizes. In this paper we investigate *restrictive compression* techniques for level 1 data cache, to avoid an increase in the cache access latency. The basic technique — All Words Narrow (*AWN*) — compresses a cache block only if all the words in the cache block are of narrow size. We extend the *AWN* technique to store a few upper half-words (*AHS*) in a cache block to accommodate a small number of normal-sized words in the cache block. Further, we make the *AHS* technique adaptive, where the additional half-words space is adaptively allocated to the various cache blocks. We also propose techniques to reduce the increase in the tag space that is inevitable with compression techniques. Overall, the techniques in this paper increase the average L1 data cache capacity (in terms of the average number of valid cache blocks per cycle) by about 50%, compared to the conventional cache, with no or minimal impact on the cache access time. In addition, the techniques have the potential of reducing the average L1 data cache miss rate by about 23%.

1 Introduction

CMOS scaling trends result in faster transistors and relatively longer wire delays, making it difficult to have low latency caches [9]. This is due to the long wires required to access the RAM structures. This trend has resulted in pipelined cache access and small-sized Level 1 caches. Another important parameter that affects cache design is the energy consumption in the cache [4, 13, 22]. To reduce the cache energy consumption, designers have decoupled the tag comparisons from the data access [15]. Figure 1 shows the decoupled and pipelined cache read access [11]. A cache access starts with decoding the set index. In the next cycle, byte-offset is decoded in parallel to address tag comparisons, and the bit-lines in the data array are pre-charged. The tag comparisons control whether or not the data is read from a cache block. If the data is read, then it is then driven to the units that requested the data.

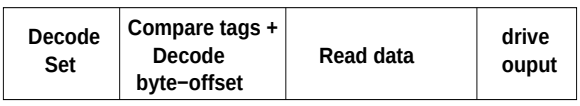

Figure 1: Pipelined Data Cache Read Access

Increasing Level 1 cache size impacts cache access time, whereas small sized caches increase the cache miss rate. Cache compression is a popular approach to increase the capacity of a small cache. However, elaborate compression techniques cannot be applied to the level 1 cache, as they increase the cache access latency. In this paper, we propose *restrictive compression* techniques, that do not increase the cache access latency, and still result in significant increase in the L1 data cache capacity.

Our basic technique — All Words Narrow (*AWN*) — compresses a cache block only if all the words in the cache block are of narrow width (requiring less than a particular number of bits for representation). *AWN* technique alone increases the level 1 data cache capacity (in terms of the average number of valid blocks per cycle) by about 20%, compared to a conventional data cache. We extend the *AWN* technique by providing additional space for a few upper half-words (*AHS*) in a cache block to accommodate a small number of normal-sized words in each cache block. *AHS* technique increases the L1 data cache capacity by almost 90% (with an average of about 50%). We also investigate an adaptive *AHS* technique — *AAHS* — where the number of additional half-words allocated to a cache block is varied depending on the requirement. Finally, we propose a technique — *OATS* — to reduce the increased cache tag space requirement, which is inevitable with any cache compression technique. It is important to note that we performed our experiments with a 32-bit architecture, whereas our techniques will be much more beneficial for a 64-bit architecture.

The rest of the paper is organized as follows. Section 2 discusses the motivation behind our techniques and the related work. Section 3 presents the *AWN* and the *AHS* cache compression techniques. Section 4 presents the experimental setup and the results. Section 5 presents the *OATS* and the *AAHS* techniques, along with their results. Section 6 presents a sensitivity study as the block size and cache size is varied. We conclude in Section 7.

2 Motivation and Related Work

2.1 Motivation

Elaborate cache compression techniques cannot be applied for L1 caches, because such techniques require multiple cycles to decompress the data read from the cache, thus increasing the cache access latency. For instance, if a cache block is compressed by ignoring all the insignificant higher order bytes, then

such a cache block will consist of both compressed as well as uncompressed words. The byte-offset of each word in the cache block will depend on the size of the words before it. This will require recalculating the byte-offset to read a word from the block, shown in Figure 2. Therefore, it is imperative that any compression technique that is applied to L1 caches should not require updates to the byte-offset. We call such compression techniques as *restrictive compression techniques*.

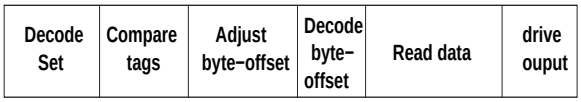

Figure 2: Pipelined DCache Read with byte-offset adjustment

To motivate the techniques proposed in this paper, we measure the number of normal-sized words (that require more than 16 bits for representation) in cache block. Figure 3 presents the results of the measurements in the form of a stacked graph, where each bar is divided into the percentage of cache blocks with 0, 1, 2, 3, 4, and more than 4 normal-sized words.

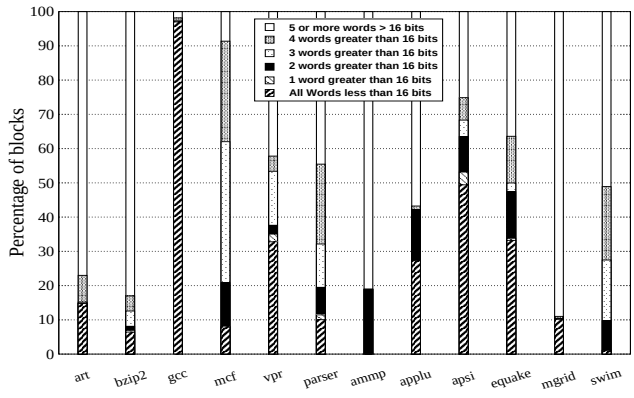

Figure 3: Percentage distribution of word sizes in the new cache blocks brought into the L1 Dcache

Figure 3 shows that, for many benchmarks, a significant fraction of the new cache blocks that are brought into the L1 data cache have all narrow words. In addition, there are many cache blocks that have just 1 or 2 normal words. Narrow words are less prevalent in the floating-point benchmarks because of the IEEE 754 format.

2.2 Related Work

Compression techniques have mainly been proposed to compress the data in main memory [5, 7] and in L2 cache [10, 6, 19, 20, 21, 14]. More sophisticated compression techniques (such as XRL compression [17], and parallel compression with dictionary construction [12]) can be applied to main memory and L2 cache as they are more tolerant to increase in hit times.

Level 1 data cache compression techniques have been proposed in [16, 24, 25]. However, all these techniques increase the L1 access time. Yang et. al. [24, 25] use a small Frequent Value Cache (FVC) to store small number of frequently seen values in cache blocks. They then use a separate buffer to store the masks for the words in a cache block. A level 1 data cache access will read the mask at the byte-offset specified by the memory reference, and then based on the decoding of the mask either the frequent value cache is accessed or the data array is accessed. This technique will increase the cache access latency by at least 1 cycle.

Kim et. al [16] compress the higher order "insignificant" bits of words in a cache block to save energy. The higher order significant bits and the lower order bits of the words are stored in separate cache banks. The bank storing the higher order bits is accessed if required, doubling the cache access latency for uncompressed words. Other techniques [4, 2] also compress the higher order "insignificant" bits of words to save energy in caches.

Exploiting narrow width operands for power and performance improvements, in hardware modules other than the cache, has also been proposed in [1, 3, 18].

3 Restrictive Compression Techniques

We consider a word narrow if it can be represented using 16 bits, and can be reconstructed by *sign-extending* the narrow word.

3.1 All Words Narrow (AWN)

In this technique, a cache block is compressed only if *all the words* in the cache block are of narrow size. We call such a cache block a *narrow cache block*, and a cache block where all the words are represented using the entire set of bits as a *normal cache block*. We call the physical RAM space provided in the cache for a cache block as a *physical cache block*. A *physical cache block* can hold one *normal cache block* or up to two *narrow cache blocks*, as shown in Figure 4(a). Note that, the last word in the lower cache block is not narrow because sign-extension of 9401 will result in $ffff9401$, whereas this is not the case with the last word in the upper block 7401.

Figure 4(b) shows that each *physical cache block* is provided with additional tag space and status bits for the additional *narrow cache block* (as is the case for all compression techniques), and a *width bit* signifying the presence of a *normal cache block* (width bit = "0") or two *narrow cache blocks* (width bit = "1"). In parallel to tag comparisons, the byte-offset is decoded, by also considering the *width bit* for the cache block. The *size bits*, also used in a conventional cache, and the *width bit* decide the number and the location of bit-lines to be activated. When reading a word, if the *width bit* is set, 2 half-words are simultaneously read from two locations in the cache block (by considering the byte-offset as a byte-offset as well as a half-byte offset). If the *width bit* is reset, 1 normal-sized word is read from only one location as in the conventional case. Bit-line activation for load half-word and load byte instructions will also be accordingly modified. Figure 5 illustrates read accesses of a 32-bit word if the *width bit* is set or reset. Hence, the *AWN* technique avoids byte-offset updates. The data read from the caches has to be sign-extended if the *width bit* is set.

The replacement policy used in the *AWN* technique is still the LRU policy (with some modifications). If the new cache block brought in is a *narrow cache block*, then the conventional LRU policy is followed. Note that the least recently used cache block could be a *narrow* or a *normal* cache block, and is replaced accordingly. When a *normal cache block* is brought in, the age of the most recently used cache block in a *physical cache block*

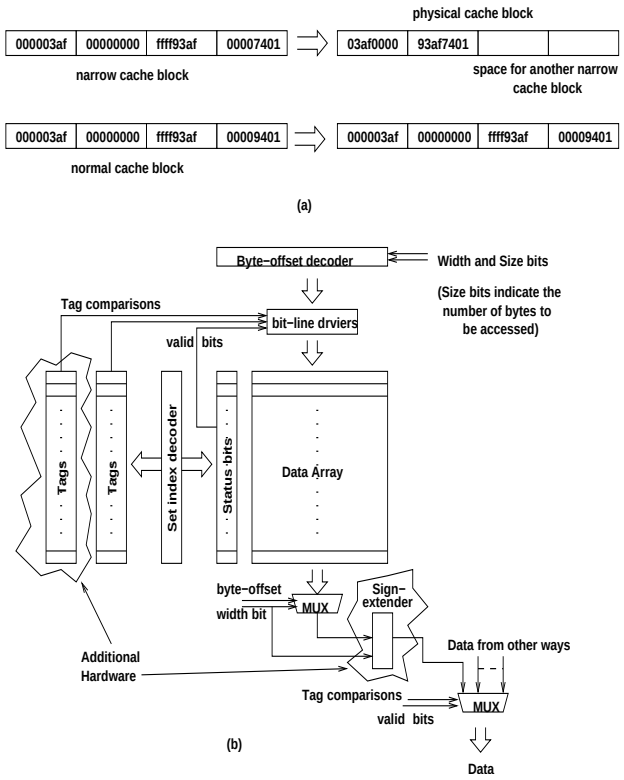

Figure 4: (a) Placement of narrow and normal cache blocks in a physical cache block; (b) Schematic data cache read access in the *AWN* technique

is chosen for comparison to find the least recently used cache block. For instance, if a *physical cache block* has 2 *narrow cache blocks*, and one of them is the most recently used, then the cache blocks in that *physical cache block* will not be replaced. This ensures that the *AWN* technique will not perform worse than the conventional cache.

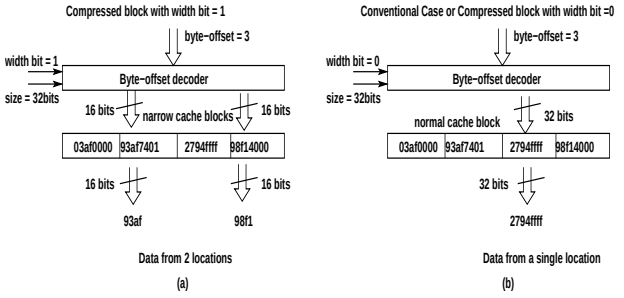

Figure 5: Read access of a 32-bit word when (a) the width bit is set; and (b) the width bit is reset

The write operation (for a store instruction) is performed similarly, after checking the size of the data to be written. A potential problem could arise if a write operation writes a normal value into a *narrow cache block*. We observed that such cases are very rare (on an average, about 2%). In our implementation, if a write operation writes a normal value into a *narrow cache block*, the *narrow cache block* is read and converted into a *normal cache block*. It then replaces an existing cache block based on the modified LRU policy discussed earlier. In our measurements, we consider such cases as cache misses.

3.2 Additional Half-Word Storage (AHS)

In the *AWN* technique, even a single normal-sized word will make the entire cache block a *normal cache block*, thus not benefiting from the technique. To make the cache blocks with a few normal-sized words as *narrow cache blocks*, we provide additional half-words storage for each *physical cache block*, to store the higher order half-words of the few normal-sized words. In the *AHS* technique, we equally divide the additional half-words among all the *narrow cache blocks* in a *physical cache block*. Hence, if a *physical cache block* is provided with 2 additional half-words, then that cache block can contain two *narrow cache blocks*, each with a maximum of one normal-sized word, but it can not hold one *narrow cache block* with 2 normal-sized words. In the *AHS* scheme, an *extra storage* bit (for the case of 2 half-words per *physical cache block*) is provided for each half-word in the *physical cache block*, indicating the presence of an upper half-word in the additional storage. Figure 6(a) shows how data is packed in a physical cache block. The lower cache block is a normal cache block, which gets converted into a narrow one by placing the upper half-word “01f0” for the word “01f09401” in the additional storage, and set the corresponding *extra storage* bit. Figure 6(b) shows the schematic data read access from a cache with *AHS*. To read data from a cache with the *AHS* technique, once the byte-offset is decoded, the *extra storage* bit is read to determine whether the additional half-words need to be read or not. Alternately, additional storage for a cache block can always be read on a tag comparison. The additional delay in reading the additional half-words can easily be overlapped with reading the data from the data array (based on our experiments with cacti [23]). The rest of the data cache access will remain the same as in the *AWN* technique.

Cache Access Latency Impact: The *AWN* technique will incur additional delay in the byte-offset decoder (it also considers the *width* bit) and the *output data* stage (because of additional control signals in the MUXes and a *sign-extender* required between the 2 levels of MUXes in Figure 4(b)). The *AHS* technique incurs additional delay in the byte-offset decoder for reading the *extra storage bit*. However, these additional delays will not impact the overall cache access latency. Our experiments with the cacti tool [23] show that the delay in the *read data* stage is more than double the delay in the *output data stage* and the *byte-offset* decoder, and that the additional delays can be easily absorbed in the shorter *output data* and *decode byte-offset* pipeline stages. We performed our experiments with a 0.18μm technology.

Cache Energy Consumption In the *AWN* and the *AHS* techniques, additional energy is primarily consumed in the additional tag comparisons, and in the L1 cache data array as a result of an increase in the hit rate. However, on a hit, the number of bits that the *AWN* technique reads from the data array is almost the same as that read by the conventional cache. For instance in a load word, the *AWN* technique reads either 2 half-words or 1 word. Some minimal additional energy will also be consumed in maintaining and reading the additional status and width bits. In

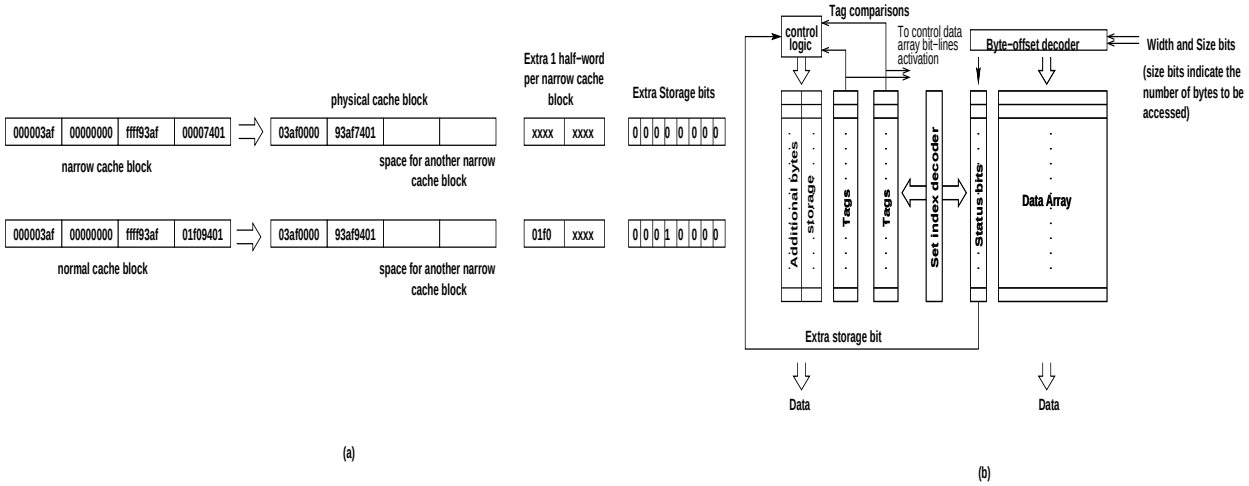

Figure 6: (a) Cache block packing with *AHS* technique; (b) Schematic data cache read access in *AHS* technique

section 5, we propose a technique to reduce the additional complexity and energy consumption introduced by the additional address tags. On the other hand, because of a reduction in the L1 data cache miss rate, energy consumption in the L2 cache and in the bus between the L2 and the L1 data caches will reduce. Overall, the techniques discussed in this paper can result in savings in the total energy consumption in the L1 and the L2 data caches and the bus between the L1 and L2 data caches. However, performing a detailed energy consumption analysis of the schemes is beyond the scope of this paper.

4 Performance Results

4.1 Experimental Setup

We use a modified SimpleScalar simulator [8], simulating a 32-bit PISA architecture[1]. The hardware features and default parameters that we use are given in Table 1. For benchmarks, we use a collection of 6 integer (`vpr`, `mcf`, `parser`, `bzip2`, `gcc`, and `art`), and 6 FP (`equake`, `mgrid`, `swim`, `applu`, `ammp`, and `apsi`) benchmarks, using *ref* inputs, from the SPEC2K benchmark suite. The statistics are collected for 500M instructions after skipping the first 1B instructions. Even though the schemes in the paper can be applied to any cache, we employ the schemes only for the Level-1 data caches, where each *physical cache block* can hold up to 2 *narrow cache blocks*.

4.2 Results

In this section, we present the L1 data cache capacity (in terms of average number of valid cache blocks present in the cache per cycle) results with the *AWN* and the *AHS* techniques. Figure 7 shows the percentage increase in cache capacity for the 8KB 2-way, 32-byte block, cache with *AWN* and *AHS* techniques, compared to the conventional 8KB 2-way cache. Figure 7 shows that the percentage increase in capacity increases from *AWN* to *AHS* with 2 additional half-words (*AHS-2*)to *AHS* with 4 additional half-words (*AHS-4*). *AWN* technique increases the cache capacity by about 20% with minimal additional storage and no access time impact. *AHS-4* technique increases the cache capacity by about 50% on an average, with about 38% additional storage and minimal, if any, impact on cache access time. It is important to note that for the *AHS* techniques, in some benchmarks (such as `applu` and `bzip2`), the increase in cache capacity may be less than the space overhead of our techniques. If the space overhead required for the *AHS* technique is utilized towards making the cache bigger, the overall cache access time can be severely impacted. However, in our techniques, the cache capacity in increased without any increase in the overall cache access time. In addition, the *OATS* technique in Section 5 helps in reducing the space overhead of the *AHS* techniques.

[1] The techniques of this paper will be much more beneficial for a 64-bit architecture.

5 Enhanced Techniques

5.1 Optimizing Address Tags (OATS)

As is the case with all the cache compression techniques, additional tag space and tag comparisons are required for the *AWN* and the AHS techniques. In this section, we propose a technique to reduce the additional tag space and tag comparisons. Intuitively, the higher order bits of the address tags in a set are expected to be the same. Our experiments confirmed the suggestion. Hence, in the optimizing address tags (OATS) technique, instead of providing the entire set of bits used for the address tag, only a small number of additional tag bits are provided for each *physical cache block*. For instance, a *physical cache block* with 22 bit address tags in the conventional cache, may be provided with 24 tag bits in the *AWN* and the *AHS* techniques, partitioned into three parts; one of 20 higher order bits, and 2 parts of 2 lower order bits each. In this case, the *physical cache block* can either hold just 1 cache block or it can hold two *narrow cache blocks* that have the same 20 higher order tag bits. Tag comparisons are performed accordingly. When compared to a conventional cache with 22 tag bits, the cache with the *OATS* technique requires just 2 additional tag bits and tag bit comparisons. Note that the *OATS* technique reduces the space overhead of the *AHS-4* technique to about 30%. The cache capacity results with the *OATS* technique are presented in the Section 5.3.

Parameter	Value	Parameter	Value
Fetch/Commit Width	8 instructions	*Instr. Window Size*	96 Int/64 FP instructions
ROB Size	256 instructions	*Frontend Stages*	9 pipeline stages
Unified Phy. Register File	128 Int/128 FP, 2-cycle pipelined access.	*Int. Functional units*	3 ALU, 1 Mul/Div, 2 AGUs
Issue Width	5 Int/ 3 FP	*FP Functional Units*	3 ALU, 1 Mul/Div
Branch Predictor	gshare 4K entries	*BTB Size*	4096 entries, 4-way assoc.
L1 - I-cache	16K, direct-mapped, 2 cycle latency	*L1 - D-cache*	8KB, 2-way assoc., 32 bytes block 4 cycle pipelined access
Memory Latency	100 cycles first chunk 2 cycles/inter-chunk	*L2 - cache*	unified 512K, 8-way assoc., 10 cycles

Table 1: Default Parameters for the Experimental Evaluation

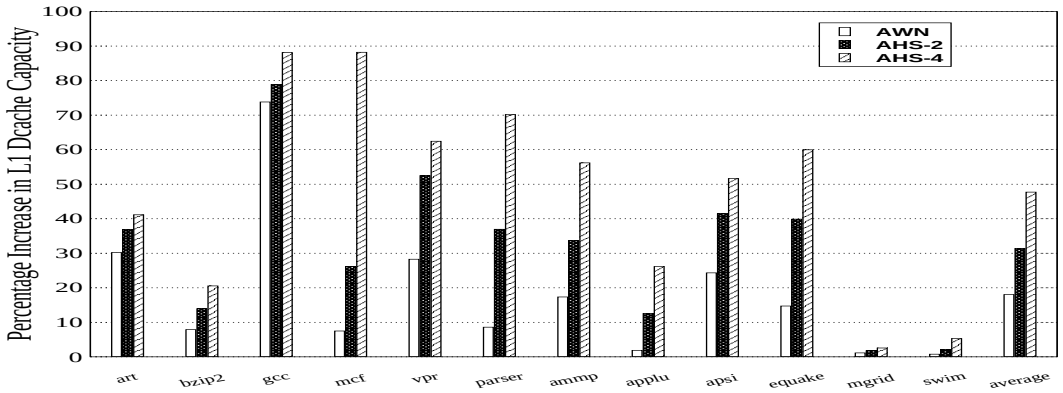

Figure 7: Percentage increase in L1 data cache capacity wrt a 2-way 8KB conventional cache

5.2 Adaptive AHS (AAHS)

In the *AHS* technique discussed in Section 3.2, the additional half-words storage provided with each *physical cache block* is not optimally utilized. For instance, if space is provided to store the upper halves of 4 words, the space is equally divided among the 2 potential *narrow cache blocks* that can occupy the *physical cache block*. In this case, the *physical cache block* cannot contain one block with 1 *normal-sized* word and another cache block with 3 *normal-sized* words. Here, we propose an adaptive scheme that allows the narrow cache blocks to use varied number of additional half-words. If 2 additional half-words are provided, then in the *AAHS* technique, 2 *extra storage* bits are required, because a cache block can use both the half-words, and the *extra storage* bits will be *01* and *10*. To avoid increasing the number of *extra storage* bits required for 4 additional half-words, we restrict the maximum number of half-words that a cache block can use to 3. Hence, *extra storage* bits "01", "10", and "11" correspond to 1^{st}, 2^{nd}, and 3^{rd} half-words for the first *narrow cache block* and 2^{nd}, 3^{rd}, and 4^{th} half-words for the second *narrow cache block*. Figure 8 shows the contents of a *physical cache block* containing 2 *narrow cache blocks*. The 1^{st} block has 1 normal word ("00ff7401") and the 2^{nd} block has 3 normal words ("01003801", "00100000", and "00009401").

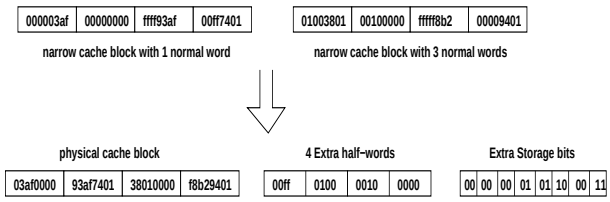

Figure 8: Converting 2 normal cache blocks into narrow cache blocks and packing them in the *AAHS* technique

5.3 Results

In this section, we present the L1 data cache results with the *OATS* and the *AAHS* techniques, in Figure 9, respectively, compared to the conventional 8KB 2-way conventional L1 data cache. For the *OATS* technique, we experiment with providing additional 2 (*OATS-2*)and 4 (*OATS-4*) bits for the additional tag, using the *AHS-4* technique words as the base case. For the *AAHS* technique, we experiment with 2 (*AAHS-2*), and 4 (*AAHS-4*)additional half-words per *physical cache block*. As seen in Figure 9, providing just 2 additional tag bits per *physical cache block* is enough to capture almost all the *narrow cache blocks* obtained with all the bits provided for the entire tag. Figure 9 shows that the *AAHS-4* technique increases the percentage increase in cache capacity, compared to the conventional cache, by about 50%, which is only slightly better than *AHS-4* technique. However, *AAHS-2* performs about 8% better than *AHS-2*.

Our experiments showed that the increase in cache capacity did not translate into comparable reductions in cache miss rates. On an average, the L1 data cache miss rate reduced by about 1%, 3%, and 7% for the *AWN*, *AHS-2*, and *AHS-4* techniques, respectively. The small reduction in cache miss rate is because different benchmarks require different cache sizes, and if the

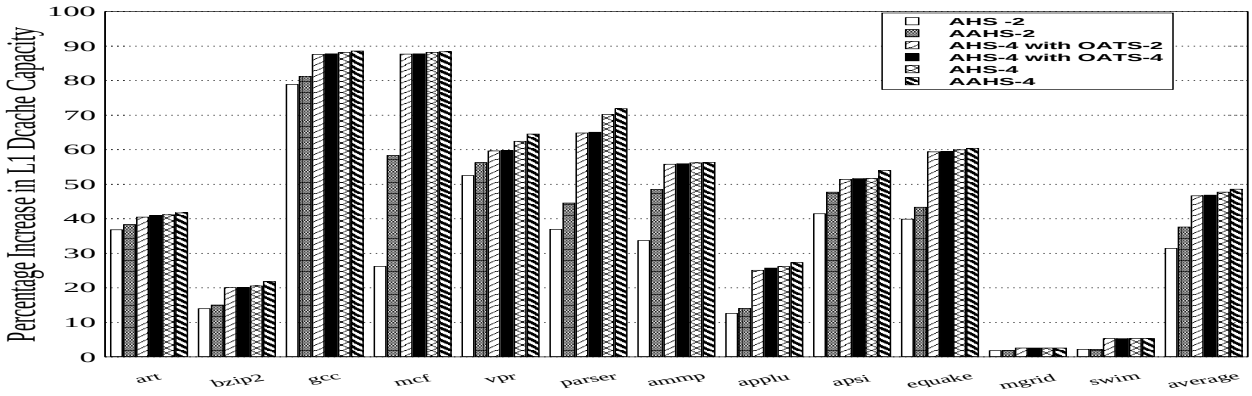

Figure 9: Percentage increase in L1 Dcache capacity, compared to a 2-way 8KB conventional cache

8KB 2-way cache is enough to get rid of most of the misses, then increasing the cache capacity may not impact the miss rate considerably. To study the potential of our techniques in reducing the L1 data cache miss rate, we experiment with different cache sizes for different benchmarks. The cache size is chosen for a benchmark such that a noticeably reduction in cache miss rate is observed when going from a 2-way smaller conventional cache to a 4-way larger conventional cache (double the size of the 2-way cache). We also experimented with a conventional cache provided with 38% more data space (equal to the overhead of the *AAHS-4* technique), where some sets had 2 cache blocks and some had 3 cache blocks. Figure 10 presents the percentage miss rate reduction for *AHS-4*, AHS-4 with OATS-4, and *AAHS-4* techniques on a 2-way smaller cache, compared with a 38% more conventional cache. Figure 10 shows that an average miss rate reduction of about 23% can be achieved with the *AAHS-4* technique, whereas, the 38% more cache achieves a reduction of about ..%. However, for some benchmarks, our techniques perform significantly better than the 38% more conventional cache. Note that the space overhead of the *AAHS-4* technique can be further reduced by using the *OATS* technique.

6 Sensitivity Study

In this section, we measure the increase in the L1 data cache capacity as the block size is varied from 16 bytes to 128 bytes, while keeping the cache size constant. Figure 11 presents the measurements for the various cache block sizes for the *AHS-4* technique.

As seen in Figure 11, the percentage increase in the L1 data cache capacity generally reduces as the block sizes are increased from 16 bytes to 128 bytes. This is expected, because the number of *narrow cache blocks* reduce as the block size is increased. Note that, as the block size increases, the percentage overhead of the techniques discussed in this paper also reduces. Nevertheless, the percentage increase in the L1 data capacity rate is still about 38% for a block size of 64 bytes, and about 27% for a block size of 128 bytes.

7 Conclusion

With the CMOS scaling trends and slow scaling of wires as compared to the transistors, the cache access latencies will increase in the future microprocessors. To prevent the increasing latencies from affecting the cache throughput, the L1 caches are small-sized and their accesses are pipelined. Small-sized L1 data caches can result in significant performance degradation due to increased miss rates. Compression techniques can be used to increase the L1 data cache capacity. However, these compression techniques cannot alter the byte-offset of the memory reference, to avoid any increase in the cache access latency.

In this paper, we propose *restrictive* cache compression techniques that do not require updates to the byte-offset, and hence result in minimal, if any, cache access latency impact. Our basic technique — *AWN* — compresses a cache block only if all the words in the cache block are of small size. The compressed cache blocks are then packed together in a single *physical cache block*. The *AWN* technique requires minimal additional storage in the cache and results in a 20% increase in the cache capacity. We extend the *AWN* technique by providing some additional space for the upper half-words — *AHS* — of a few *normal-sized* words in a cache block, with an aim to convert them into narrow blocks. We further extend the *AHS* technique — *AAHS* — so that the number of upper half-words used by a cache block can vary depending on the requirement. *AHS* and *AAHS* techniques increase the cache capacity by about 50%, while incurring a 38% increase in the storage space required, compared to a conventional cache. However, these techniques still do not impact the cache access latency. To reduce the additional tag requirements (which is inevitable with any cache compression technique), we propose reducing the number of additional tag bits provided for the additional cache blocks to be packed in a *physical cache block*, reducing the overhead of the *AHS* techniques to about 30%. Our studies showed that providing just 2 additional tag bits is enough to give a performance almost equal to that obtained with doubling the number of tag bits.

References

[1] D. Brooks and M. Martonosi, "Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance," *Proc. HPCA*, 1999.

[2] M. Ghosh, et al., "CoolPression —A Hybrid Signifi cance Compression Technique for Reducing Energy in Caches." *Proc. IEEE Int'l SOC Conference*, 2004.

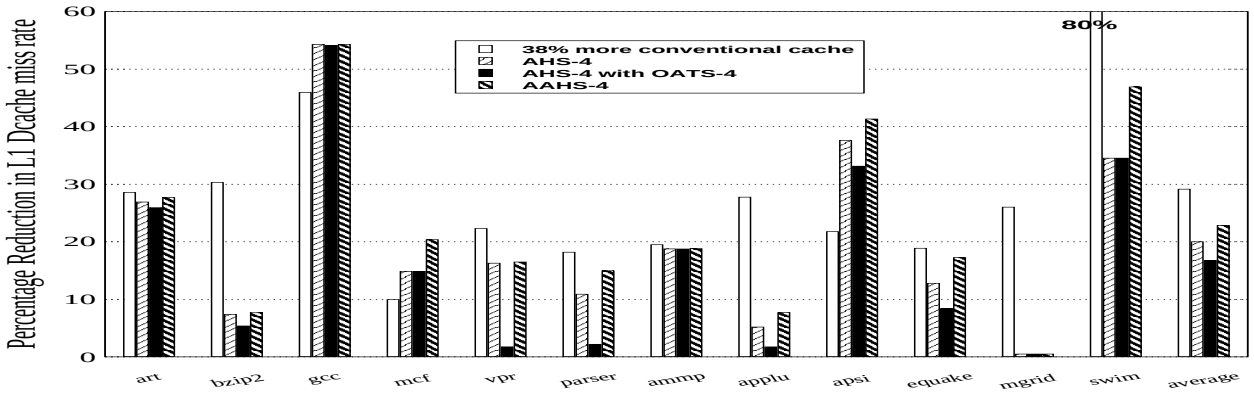

Figure 10: Percentage reduction in L1 Dcache miss rate for different sized caches for different benchmarks, compared to a 2-way smaller conventional cache

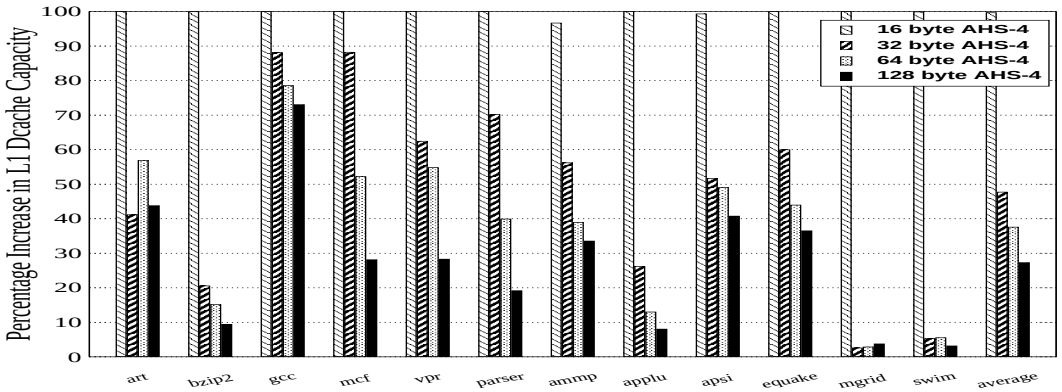

Figure 11: Percentage increase in L1 Dcache capacity, compared to a 2-way 8KB conventional cache; with varying cache block sizes

[3] G. Loh, "Exploiting data-width locality to increase superscalar execution bandwidth," *Proc. Micro-35*, 2002.

[4] L. Villa, et. al., "Dynamic zero compression for cache energy reduction," *Proc. of Micro-33*, 2000.

[5] B. Abali and H. Franke, "Operating System Support for Fast Hardware Compression of Main Memory Contents", *Workshop on Solving the Memory Wall Problem*, 2000.

[6] A. Alameldeen and D. Wood, "Adaptive Cache Compression for High-Performance Processors", *Proc. ISCA-31*, 2004.

[7] C. Benveniste, et. al., "Cache-Memory Interfaces in Compressed Memory Systems", *Workshop on Solving the Memory Wall Problem*, 2000.

[8] D. Burger and T. M. Austin, "The SimpleScalar Tool Set, Version 2.0," *Computer Arch. News*, 1997.

[9] T. Chappell, et. al., "A 2-ns cycle, 3.8-ns access 512-kB CMOS ECL SRAM with a fully pipelined architecture," *IEEE Jour. of Solid State Circuits*, 26(11):1577-1585, 1991.

[10] D. Chen, et. al., "A Dynamically Partitionable Compressed Cache", *Singapore-MIT Alliance Symposium*, 2003.

[11] Z. Chishti, and T. N. Vijaykumar, "Wire Delay is not a problem for SMT (in the near future)," *Proc. ISCA-31*, 2004.

[12] P. Franaszek, et. al., "Parallel Compression with Cooperative Dictionary Construction", *Proc. Data Compression Conference*, 1996.

[13] M. K. Gowan, et. al., "Power Considerations in the Design of the Alpha 21264 Microprocessor," *Proc. DAC*, 1998.

[14] E. Hallnor and S. Reinhardt, "A Compressed Memory Hierarchy using an Indirect Index Cache", *Technical Report CSE-TR-488-04, University of Michigan*, 2004.

[15] R. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro*, March/April 1999.

[16] N. Kim, et. al., "Low-Energy Data Cache using Sign Compression and Cache Line Bisection", *Workshop on Memory Performance Issues*, 2002.

[17] M. Kjelso, et. al., "Design and Performance of a Main Memory Hardware Data Compressor", *Proc. EUROMICRO Conference*, 1996.

[18] S. Kumar, et. al., "Bit-Sliced Datapath for Energy-Efficient High Performance Microprocessors", *Workshop on Power-Aware Computer Systems(PACS'04)*, 2004.

[19] J. Lee, et. al., "Design and Evaluation of a Selective Compressed Memory System", *Proc. ICCD*, 1999.

[20] J. Lee, et. al., "An On-chip Cache Compression Technique to Reduce Decompression Overhead and Design Complexity", *Journal of Systems Architecture: The EUROMICRO Journal, 46(15):1365-1382*, 2000.

[21] J. Lee, et. al., "Adaptive Methods to Minimize Decompression Overhead for Compressed On-chip Cache", *International Journal of Computers and Application, 25(2)*, 2003.

[22] S. Manne, et. al., "Pipeline Gating: Speculation Control For Energy Reduction," *Proc. ISCA*, 1998.

[23] P. Shivakumar, and N. Jouppi, "CACTI 3.0: An Integrated Cache Timing Power, and Area Model," *Technical Report, DEC Western Research Lab*, 2002.

[24] J. Yang, et. al., "Frequent Value Compression in Data Caches", *Proc. Micro-33*, 2000.

[25] Y. Zhang, et. al., "Frequent Value Locality and Value Centric Data Cache Design", *Proc. ASPLOS*, 2000.