POSTER: Variable Sized Cache-Block Compaction

Sayantan Ray, *Student Member, IEEE*
Department of Computer Science and Engineering
Indian Institute of Technology Madras, Chennai, India
raysayantan07@gmail.com

Madhu Mutyam, *Senior Member, IEEE*
Department of Computer Science and Engineering
Indian Institute of Technology Madras, Chennai, India
madhu@cse.iitm.ac.in

***Abstract*—Data blocks compressed to different sizes can be stored together inside a single cache-block to increase space utilization. However, the lack of a common *size offset* makes it challenging to locate individual blocks without additional tag overhead. We propose *Variable Sized Cache-Block Compaction* (VSCC) that allows us to store variable sized compressed blocks together and locate them inside a cache-block by using their compression encodings – available inside the tag metadata. We introduce a novel read/write scheme and a new BDI compression encoding, which reduce the necessary operations by 50%. Experimental results reveal that VSCC outperforms state-of-the-art techniques from the performance and energy point of view while keeping the storage overheads within acceptable limits.**

***Keywords*-cache, compaction, energy efficiency, performance.**

I. Introduction

Compression exploits recurring patterns in data and represents it using fewer bits. Consequently, the space saved inside a cache-block can be utilized by storing additional compressed blocks. A *compaction* scheme tracks multiple compressed blocks of data stored into a single cache-block with minimal storage overhead. Recent compaction techniques [1], [2] have adopted *sectoring* [3] – which uses a single tag to track multiple neighbouring blocks (collectively known as *sectors)* with some additional metadata. Uniformly compressed neighboring data stored inside a cache-block can be tracked using a common *size offset*. However, if two such blocks are compressed to variable sizes, we can no longer maintain such an offset to determine their locations. *Decoupled compressed cache* (DCC) [1] stored variable-sized blocks together with the help of additional tag structures, while *yet another compressed cache* (YACC) [2] compromised on the overall performance to eliminate the storage overhead. We propose *Variable Sized Cache-Block Compaction* (VSCC), which compacts neighboring blocks of variable sizes inside a cache-block without a significant storage overhead. VSCC locates a compressed block by using the compression encodings of the pre-existing adjacent compressed data. This eliminates the need for any additional tag structures. VSCC strikes the right balance between high cache utilization and low storage overheads and provides better system performance at lower energy cost as compared to previous works.

II. Motivation

Storing uniformly sized compressed blocks inside a cache-block to have a common *size offset* restricts us from fitting variable-sized neighboring blocks into a cache-block even if there is sufficient space. This leads to internal fragmentation. Previous works have used additional tag structures to identify the location of individual compressed blocks of variable sizes inside a cache block. This adds to the access latency, storage overhead, and energy consumption of the cache.

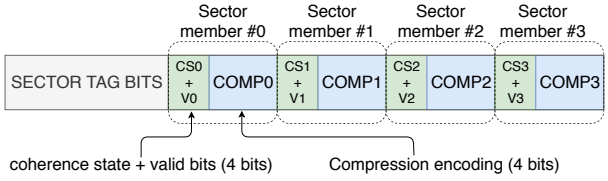


Fig. 1: Tag Structure of $VSCC_{S4}$.

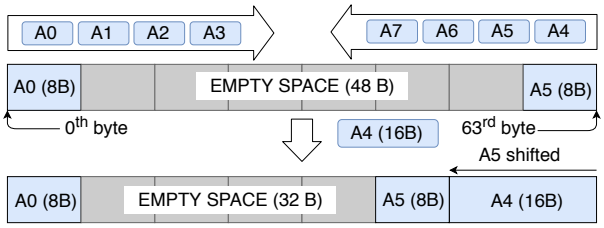


Fig. 2: Read/Write Scheme.

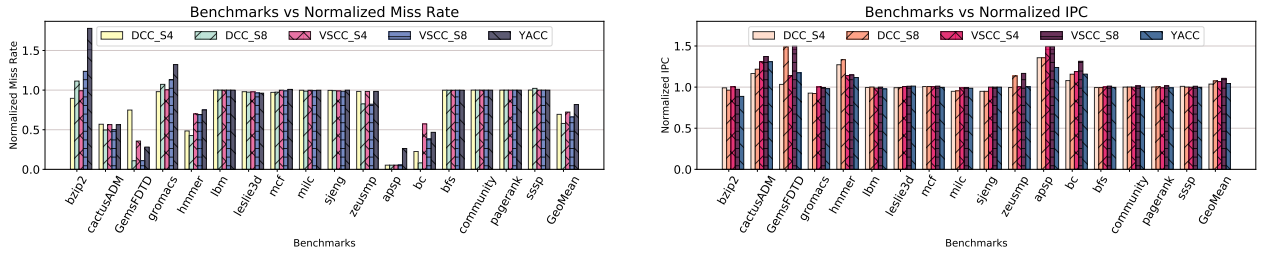
III. Variable Sized Cache-Block Compaction

VSCC compacts neighboring blocks of variable sizes inside a cache-block, as long as there is sufficient space. The starting byte of a block is obtained by adding the compression sizes of other blocks present inside the cache-block preceding the requested block in order. As the compression sizes are already available inside the tag itself, VSCC does not require any additional tag structures.

Tag Structure: A sector tag tracks all the members of the corresponding sector. Figure 1 shows the tag structure of VSCC using a sector of 4 neighboring blocks (denoted as $VSCC_{S4}$). As multiple blocks share the same sector tag, a sector tag hit on a request for one of the blocks is followed by checking its corresponding valid bit. We determine the block position by adding the compression encoding of all the blocks that precede the requested block in order.

Read/Write Scheme: The first half of the sector members ($A0$-$A3$) are stored in the cache-block starting from the 0^{th} byte (left to right). Blocks belonging to the latter half of the

978-1-7281-3613-4/19/$31.00 ©2019 IEEE
DOI 10.1109/PACT.2019.00050



(a) Normalized Miss-rates.

(b) Normalized IPCs.

Fig. 3: Results for a 2MB 16-way LLC.

TABLE I: Modified BDI Encoding.

Base	Δ	Size	Code	Base	Δ	Size	Code	Base	Δ	Size	Code
1B	N/A	1B	1001	8B	N/A	8B	0001	8B	1B	16B	0010
8B	2B	24B	0011	8B	4B	40B	0101	4B	1B	20B	1011
4B	2B	36B	1101	2B	1B	34B	1101	N/A	N/A	64B	1000

TABLE II: Results Summarization (geometric mean).

	DCC_{S4}	DCC_{S8}	$VSCC_{S4}$	$VSCC_{S8}$	$YACC_{S4}$
Storage Overhead	16%	25.5%	4.8%	10.5%	1.8%
Eff. Cache Capacity	1.83	2.36	1.67	2.03	1.55
Miss-Rate Reduction	30.6%	42%	27.6%	33.8%	18.2%
IPC Improvement	7.6%	9.3%	6.7%	10.7%	4.4%
Energy Reduction	9.7%	12.8%	9.3%	13.9%	3%

sector ($A4$-$A7$) are stored from 63^{rd} byte (right to left). This reduces the number of additions required in the worst case ($A7$) for $S8$ from six to two. For $S4$, addition operations are no longer necessary. As seen in Figure 2, this technique ensures that the empty spaces inside a cache-way exist as one in the middle, which can be filled from either side.

New BDI encoding: To avoid fetching multiple compression sizes for determining the position of a block, we introduce a new base-delta-immediate (BDI) encoding (refer to Table I). We allocate 3-bit encoding to consequent multiples of 8 (8B = 001, 16B = 010, etc). Thus a compressed block with an encoding 001 means that it occupies 8B (or 1 sub-block). For a cache-block containing two compressed blocks of size 8B (001) each, the starting byte for the third compressed block can be determined by simply adding the two encodings: $001+001=010$, which corresponds to the 16^{th} byte of the cache-block. We use the 4^{th} bit (MSB) to differentiate between two compression sizes requiring the same number of sub-blocks.

IV. Experimental Results and Analysis

By considering a single-core system with 2MB 16-way LLC, we evaluate and compare VSCC with recent works DCC [1] and YACC [2]. Both DCC and YACC use BDI [4] compression scheme. We consider benchmarks from SPEC CPU 2006 [5] and CRONO [6] running on GEM5 simulator [7]. The results presented are normalized to that of a conventional cache without any compression scheme.

Effective Cache Capacity: $VSCC_{S8}$ on average, holds $2\times$ the data of a typical cache, with only 10% increase in the physical cache space. Though DCC_{S8} achieves better effective cache capacity, it incurs high storage overhead.

Miss-Rate: Holding more data in the LLC reduces capacity misses. We see from Figure 3a that DCC provides the lowest miss-rate (normalized to a conventional cache), due to DCC's ability to store a block anywhere across a set, irrespective of its size and parent sector. By storing blocks of variable sizes together, VSCC obtains a lower miss-rate than YACC.

Performance and Energy: Reduced miss-rate leads to higher Instructions Per Cycle (IPC). Figure 3b shows the normalized IPCs for all the three techniques. Irrespective of a lower miss-rate, DCC suffers from lower IPC due to a longer run-time – attributed by the accesses made to its additional tag structure. A longer run-time and additional accesses increase the overall energy consumption of DCC. Owing to a significantly lower miss-rate, VSCC outperforms YACC as well. We observe that on an average (geometric mean), VSCC provides the greatest IPC improvement, and consequently is the most energy-efficient of the three. The results are summerized in Table II.

V. Conclusion

We proposed VSCC to compact variable-sized blocks together. Compared to a typical cache, VSCC holds $2\times$ the data while YACC holds only $1.5\times$ the data. Lack of any additional tag structure allows VSCC to outperform DCC in terms of IPC and energy, in spite of DCC having lower miss-rate. Thus VSCC achieves the right balance between overall performance and the associated overheads.

References

[1] S. Sardashti and D. A. Wood, "Decoupled compressed cache: Exploiting spatial locality for energy-optimized compressed caching," in *MICRO*, 2013, pp. 62–73.

[2] S. Sardashti *et al.*, "Yet another compressed cache: A low-cost yet effective compressed cache," *TACO*, vol. 13, no. 3, pp. 27:1–27:25, 2016.

[3] A. Seznec, "Decoupled sectored caches: Conciliating low tag implementation cost," in *ISCA*, 1994, pp. 384–393.

[4] G. Pekhimenko *et al.*, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *PACT*, 2012, pp. 377–388.

[5] "SPEC CPU2006 benchmark suite," *https://www.spec.org*.

[6] M. Ahmad *et al.*, "CRONO: A benchmark suite for multithreaded graph algorithms executing on futuristic multicores," in *IISWC*, 2015, pp. 44–55.

[7] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.