Compressed cache layout aware prefetching

Niloofar Charmchi, Caroline Collange, André Seznec

Compressed cache layout aware prefetching

Niloofar Charmchi, Caroline Collange, and André Seznec
Inria, Univ Rennes, CNRS, IRISA
Rennes, France
firstname.lastname@inria.fr

***Abstract*—The speed gap between CPU and memory is impairing performance. Cache compression and hardware prefetching are two techniques that could confront this bottleneck by decreasing last level cache misses. However, compression and prefetching have positive interactions, as prefetching benefits from higher cache capacity and compression increases the effective cache size. This paper proposes *Compressed cache Layout Aware Prefetching* (CLAP) to leverage the recently proposed sector-based compressed cache layouts such as SCC or YACC to create a synergy between compressed cache and prefetching. The idea of this approach is to prefetch contiguous blocks that can be compressed and co-allocated together with the requested block on a miss access. Prefetched blocks that share storage with existing blocks do not need to evict a valid existing entry; therefore, CLAP avoids cache pollution. In order to decide the co-allocatable blocks to prefetch, we propose a compression predictor. Based on our experimental evaluations, CLAP reduces the number of cache misses by 12% and improves performance by 4% on average, comparing to a compressed cache.**

***Index Terms*—cache compression, compaction, hardware prefetching**

I. Introduction

One of the major challenges for computers is the speed gap between processor and main memory that affects the computer performance. In order to reduce this speed gap between off-chip memory and processor, on-chip memory, referred to as cache, is adopted. Since off-chip memory latency is high, finding techniques to minimize off-chip memory accesses is of essence. Methods such as cache compression and hardware prefetching can decrease the number of last level cache (LLC) misses [13], [6]. Prefetching, however, increases the workload's working set size; it can evict useful data.

Our proposed compressed cache layout aware prefetcher (CLAP) finds a trade-off between the utility of prefetched data and the cache capacity. This positive interaction between hardware prefetching and cache compression results in performance improvement. Therefore, prefetching and compression can take advantage of each other. Compression can overcome the eviction of useful data by increasing effective cache size.

In this work we rely on the potentials of sector-based compressed caches for prefetching without causing cache pollution. To the best of our knowledge, this is the first prefetching technique that takes into account the layout of a compressed cache.

This paper makes the following contributions:

- We show that a substantial portion of the cache misses in compressed caches occur in blocks that could be allocated and compacted in a super-block that is already present in the cache. These misses could be avoided at no cost in cache capacity by bringing these blocks in the cache in advance (Section III).
- Leveraging this opportunity, we propose CLAP, a unified system that benefits from prefetching and cache compression cooperatively (Section IV).
- In order to identify which blocks should be prefetched, we propose a compression predictor (Section IV).
- We evaluate the benefits of CLAP compared to a compressed cache without prefetching. Our proposed mechanism improves performance (4% on average) and reduces number of LLC misses (12% on average). Moreover, we show that our proposed strategy is also applicable to other prefetchers, such as next-line and stride (Section V).

II. Cache compression and prefetching

Cache compression is an important approach for performance enhancement in processors [1], [11], [3]. The key idea of cache compression algorithms is to obtain the benefit of larger caches, while retaining the area and power of smaller caches. Compressed caches use compression techniques [9], [7] to reduce the size of the cache blocks and compaction methods [13], [14], [12] to allocate compressed blocks together into one data entry.

Hardware prefetching is an approach to reduce the number of cache misses. Multiple approaches have been proposed for hardware prefetching, such as stride prefetcher [4], next-line prefetcher [15] and offset prefetcher [6].

Raghavendra *et al.* [10] propose prefetched blocks compaction that can only compress and compact blocks that are prefetched, whereas our work applies compression and compaction techniques on all data in last level cache and it decides prefetching based on the compressiblity of a block. Patel *et al.* [8] improve the STeMS prefetcher by taking advantage of compression, though the compressed cache does not directly benefit from prefetching.

Yet Another Compressed Cache (YACC) [13] and Skewed Compressed Cache (SCC) [12] are among the cost-effective compressed cache layouts. Their objective is to pack the contiguous blocks of a single memory super-block in a single cache location. When all blocks in the super-block are compressible, a cache entry must be allocated for the first block, but the subsequent blocks do not require any extra cache space. The objective of this study is to exploit this opportunity to enable prefetching without pollution.

A sectored cache [5] is a set-associative cache that is designed using a shared tag among contiguous cache blocks. The

YACC architecture considers a compressed cache as a sectored cache with variable number of sub-blocks per super-block depending on the compression ratio. Moreover, the address splitting of a conventional cache and a sectored/compressed cache are not the same. These two caches have different hash functions and, therefore, they generate different access patterns.

Dictionary Sharing (DISH) [7] is a compression technique that takes advantage of the YACC layout. A common compression dictionary is shared among all sub-blocks of each super-block. In the remainder of the paper, we assume that DISH is used as the compression algorithm.

III. The Compressed Cache Underutilization Problem

A. Evaluating an LLC compressed cache

We have simulated an LLC compressed cache using DISH, for 27 benchmarks, following the evaluation methodology that will be described in Section V. In these settings, cache compression yields only modest improvements (maximum of 13% reduction in number of LLC misses for *soplex* benchmark), or may even cause more misses due to having consecutive uncompressed blocks compete for entries in the same set. Indeed, the potential capacity of the compressed cache may be underutilized due to factors such as non-compressibility or non-compactability of a block, lack of spatial locality and bulk evictions. Bulk eviction, which is the focus of this work, may happen in a compressed cache as replacement is performed at the granularity of super-blocks and all sub-blocks of a compressible super-block are evicted at once. However, it will take multiple misses to refill the compressible super-block. If we had prefetched these neighbor blocks in advance, we could have avoided the extra misses on the compressed super-block.

B. Opportunities for compressed cache layout aware prefetching

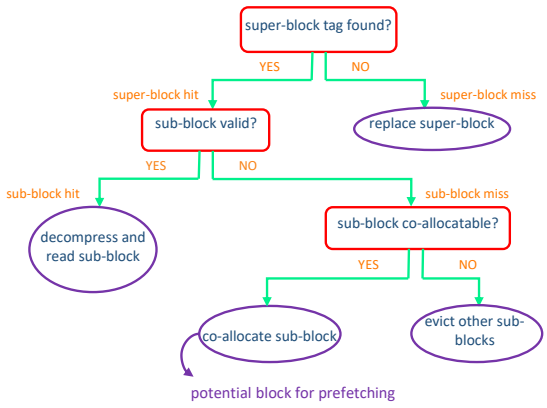


Fig. 1: Cache operations on read access

Figure 1 shows the decision diagram of compressed cache operations on an access. The cache checks all super-blocks of the set for a tag match. If no match is found, the access is a miss. Otherwise, it checks the valid bit of the corresponding sub-block for each matching entry. Finding no such valid sub-block also results in a miss.

In order to evaluate the potential of CLAP, we count the prefetching opportunities. As it is shown in Figure 1, the misses that occur on a super-block hit and sub-block miss are potential candidates for prefetching. If such block is co-allocatable with the ones that are already present in the super-block, it could have been prefetched at no expense of capacity and the miss could have been avoided. Hence, we don't need to wait for miss accesses for each sub-block to refill the compressed super-block, thereby bulk evictions become less harmful for the cache.

We measure the percentage of misses that could be avoided by taking advantage of CLAP. Figure 2 shows the percentage of LLC misses that occur on blocks that are co-allocatable within an existing super-block blocks that are already present. We could reduce the number of misses by up to 28%, by prefetching compressed blocks before they are requested by the processor, without evicting any data from the cache.

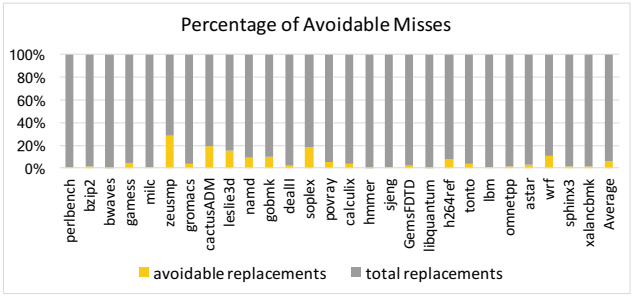


Fig. 2: Percentage of avoidable misses with CLAP

This study suggests that there is a potential for a synergistic interaction of prefetching and cache compression in processor architecture. Thus, by prefetching compressible blocks to L3, we seek the reduction in number of LLC misses.

IV. Compressed Cache Layout Aware Prefetching

In order to enhance cache compression and diminish underutilization of compressed cache, we propose CLAP, a compressed cache layout aware prefetcher. The goal is to avoid misses that happen on a co-allocatable block on a super-block hit and sub-block miss. In other words, if there is an access to a sub-block that is co-allocatable in the super-block, we could have prefetched it beforehand without any cache pollution. Thus, we could avoid this sub-block miss in the cache. In order to steer the prefetcher, we design a predictor that decides whether neighbor sub-blocks are compressible. By applying CLAP on miss accesses, we can prefetch the contiguous blocks if the block is predicted to be compressible.

A. Predicting compression

In order to find which blocks are co-allocatable and are candidate for prefetching before their content is available, we need a predictor. To achieve this goal, we implemented a compression predictor. As the predictor needs to predict the

TABLE I: Simulation parameters

Processor	ARMv8, 6-issue out-of-order at 4GHz
L1 cache	32kB, 4-way, 2 cycles
L2 cache	256kB, 8-way, 12 cycles
L3 cache	1MB, 16-way, 20 cycles
Cache line size	64-byte
Replacement policy	LRU
DRAM	DDR4 2400 MHz, bandwidth 12.8GB/s

compressibility of blocks that have not been accessed recently or never accessed, a prediction based solely on the block address is not practical. Instead, we predict the compressibility of a *stream* of memory reads, where a stream corresponds to all accesses performed by a given load instruction in the program.

The key idea of stream-based prediction is that all memory locations accessed by a given load instruction are likely to share the same data-type and similar characteristics. Thus, whether prior blocks accessed by a load instruction were compressible is a good indicator of the compressibility of future blocks loaded from the same instruction.

We use a table indexed by a hash of the PC to store the compressibility likelihood of each load instruction and update it after every read miss. The PC table has 256 sets and 4 ways, using LRU replacement policy. Each entry in this table has a 4-bit saturating counter, a compression bit and a validity bit. The saturating counter and compression bit get updated after every refill response from the memory. Whenever the block is compressible, the saturating counter gets incremented; otherwise, it gets decremented. On every access, the predictor predicts whether a block is compressible, based on the counter. Accessing to the PC table is based on each PC that is carried in the request.

B. Compressed cache layout aware prefetcher

By means of the compression predictor and taking advantage of prefetching, we proposed a compressed cache layout aware prefetcher. Whenever a read request causes a super-block miss, the compression predictor is invoked. Based on the predictor's decision, the prefetcher pre-fetches either three blocks or no block: If it predicts the block is compressible, the cache sends three requests to prefetch the three neighbor sub-blocks in a super-block. In case the predictor predicts the block is not compressible, the system does not prefetch anything.

V. Evaluation

We have implemented the DISH compression algorithm in the last level cache within the gem5 simulator [2]. We use SPEC CPU 2006 benchmarks. The statistics for the benchmarks are collected for 100M simulated instructions after a warm-up period of 50M instructions for 15 different execution snapshots. Table I shows the baseline configuration parameters with a conventional uncompressed cache. We consider a cache slice of 1MB, corresponding to the L3 cache share of a single core. Caches are non-inclusive, non-exclusive and have a write-back policy.

As it is illustrated in Figure 3, CLAP outperforms compressed cache. On average, it improves the performance by 4% comparing to compressed cache. Furthermore, CLAP reduces LLC misses of all applications by 12% on average.

A. Compression predictor

In order to implement CLAP, we propose a predictor to predict whether a sub-block is compressible. We have implemented an ideal compaction predictor which knows the future by accessing the memory beforehand to see whether the neighbor sub-blocks are co-allocatable. It predicts compactability based on the content of individual blocks, using their addresses and it only considers super-blocks that have 4 sub-blocks. However, our compression predictor prefetches all compressible blocks, regardless of compactability, which allows taking into account super-blocks that can hold 2, 3 or 4 sub-blocks.

The number of LLC misses obtained using our proposed compression predictor is within 3% on average (21% in the worst case for *zeusmp* benchmark) of the ideal compaction predictor. Therefore, our compression predictor is accurate enough to be employed to CLAP. Hence, we can prefetch even the blocks that are not co-allocatable (but only compressible), without increasing the number of misses, and they will be useful in the program.

B. Effective cache capacity

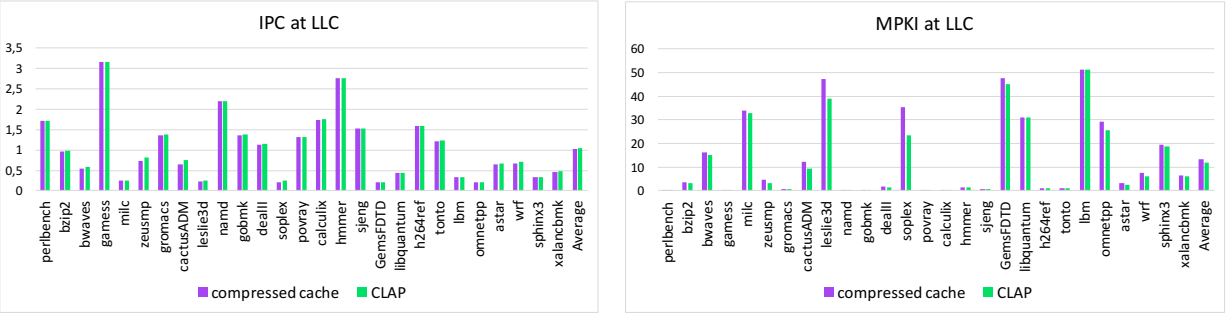
Cache compression increases the effective cache capacity, which can also be increased by taking advantage of prefetching. The effective cache capacity metric is defined as the average number of valid sub-blocks per valid super-block. Figure 4 shows the effective cache capacity for the compressed cache (DISH) and CLAP.

C. Interactions with other prefetchers

Our approach is orthogonal to other prefetching methods. We have employed the CLAP strategy to stride and next-line. Figure 5 shows the comparison between a compressed cache with stride prefetcher and a compressed cache with CLAP. In this configuration of CLAP, we do stride prefetching (with prefetch degree 16) on every cache access and when the compression predictor predicts a block is compressible, we prefetch three neighbor sub-blocks in the same super-block. Using this configuration, we manage to reduce the number of LLC misses by up to 31%. Moreover, we can take benefit of CLAP in a system with next-line prefetching and reduce LLC misses up to 29%.

D. Hardware overhead

The overhead of CLAP over the baseline prefetcher is the compression predictor table that is addressed by PC. The table has 256×4 entries and each entry has 46 bits (40-bit PC tag, 4-bit saturating counter, 1-bit compression bit, 1-bit validity bit). Hence, the table requires less than 6kB of memory, which is 0.56% of the 1MB LLC slice we consider. Increasing the cache size by this amount is unlikely to affect performance.



(a) Instructions per cycle (IPC)

(b) LLC misses per 1000 instructions (MPKI)

Fig. 3: Comparison of CLAP against compressed cache without prefetching

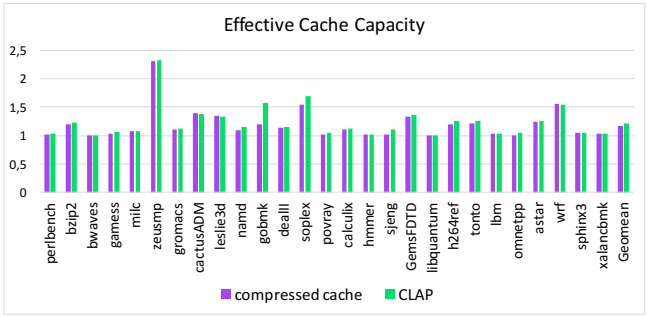


Fig. 4: Effective cache capacity relative to an uncompressed cache

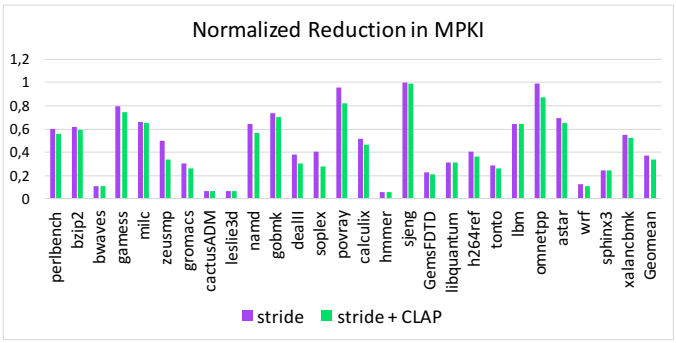


Fig. 5: Comparison of compressed cache with only stride prefetcher vs. stride and CLAP, as the number of LLC misses normalized to compressed cache without prefetching

VI. Conclusion

This paper proposes CLAP, a hardware prefetcher that is adapted to sector-based compressed cache layouts. Such compressed caches that packs neighbor blocks together are subject to underutilization; therefore, there is a potential to enhance cache compression in systems. For this purpose, we proposed a compression predictor that predicts whether neighbor sub-blocks will be co-allocatable in the same super-block. By applying CLAP on miss accesses, we can prefetch the contiguous blocks if the block is predicted to be compressible. Based on our experiments, CLAP outperforms compressed cache in terms of MPKI (up to 33%) and IPC (up to 24%). Furthermore, we show that our proposed technique can be added to other prefetchers such as stride and next-line.

VII. Acknowledgements

This project was partially supported by Région Bretagne and an Intel Research Grant. We would like to thank Daniel R. Carvalho for his help and also the reviewers for their insightful comments.

References

[1] A. R Alameldeen and D. A Wood. Adaptive cache compression for high-performance processors. *ACM SIGARCH Computer Architecture News*, 32(2):212, 2004.

[2] N. Binkert et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[3] X. Chen, L. Yang, R. P Dick, L. Shang, and H. Lekatsas. C-pack: A high-performance microprocessor cache compression algorithm. *IEEE transactions on very large scale integration (VLSI) systems*, 18(8):1196–1208, 2009.

[4] J. WC Fu, J. H Patel, and B. L Janssens. Stride directed prefetching in scalar processors. *ACM SIGMICRO Newsletter*, 23(1-2):102–110, 1992.

[5] J. S. Liptay. Structural aspects of the system/360 model 85, ii: The cache. *IBM Systems Journal*, 7(1):15–21, 1968.

[6] P. Michaud. Best-offset hardware prefetching. In *HPCA*, pages 469–480. IEEE, 2016.

[7] B. Panda and A. Seznec. Dictionary sharing: An efficient cache compression scheme for compressed caches. In *MICRO*, pages 1–12. IEEE, 2016.

[8] B. Patel, N. Hardavellas, and G. Memik. Scp: Synergistic cache compression and prefetching. In *ICCD*, pages 164–171. IEEE, 2015.

[9] G. Pekhimenko et al. Base-delta-immediate compression: practical data compression for on-chip caches. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pages 377–388. ACM, 2012.

[10] K. Raghavendra, B. Panda, and M. Mutyam. Pbc: Prefetched blocks compaction. *IEEE Transactions on Computers*, 65(8):2534–2547, 2016.

[11] S. Sardashti, A. Arelakis, P. Stenström, and D. A Wood. A primer on compression in the memory hierarchy. *Synthesis Lectures on Computer Architecture*, 10(5):1–86, 2015.

[12] S. Sardashti, A. Seznec, and D. A Wood. Skewed compressed caches. In *MICRO*, pages 331–342. IEEE Computer Society, 2014.

[13] S. Sardashti, A. Seznec, and D. A Wood. Yet another compressed cache: A low-cost yet effective compressed cache. *TACO*, 13(3):27, 2016.

[14] S. Sardashti and D. A Wood. Decoupled compressed cache: Exploiting spatial locality for energy-optimized compressed caching. In *MICRO*, pages 62–73. ACM, 2013.

[15] A. J Smith. Cache memories. *ACM Computing Surveys (CSUR)*, 14(3):473–530, 1982.