Hybrid-Comp: A Criticality-Aware Compressed Last-Level Cache

Amin Jadidi[†] Mohammad Arjomand[‡] Mahmut T. Kandemir[†] Chita R. Das[†]
[†]School of Electrical Engineering and Computer Science, Pennsylvania State University, USA
[‡]School of Computer and Electrical Engineering, Georgia Institute of Technology, USA
Email: [†]{axj945,kandemir,das}@cse.psu.edu [‡]{marjomand3}@gatech.edu

***Abstract*—Cache compression is a promising technique to increase on-chip cache capacity and to decrease off-chip bandwidth usage. While prior compression techniques always consider a trade-off between compression ratio and decompression latency, they are oblivious to the variation in criticality of different cache blocks. In multi-core processors, last-level cache (LLC) is logically shared but physically distributed among cores. In this work, we demonstrate that, cache blocks within such non-uniform architecture exhibit different sensitivity to the access latency. Owing to this behavior, we propose a criticality-aware compressed LLC that favors lower latency over higher capacity based on the criticality of the data blocks. Based on our studies on a 16-core processor with 4MB LLC, our proposed criticality-aware mechanism improves the system performance comparable to that of with an 8MB uncompressed LLC.**

I. Introduction

Despite considerable research in the past three decades leading to multi-fold improvements in cache efficiencies, the problem has become more challenging in current and future generation of processors. Workloads in the next generation of computing systems are expected to be highly data-intensive. The processing power is also steadily increasing and major manufacturers are planning to integrate hundreds of cores on a die. In such multi-core systems, computer architects employ high capacity on-chip cache hierarchies to reduce data access latency. The decision of how large to make a given cache involves trade-offs: while larger caches often reduce number of cache misses, this potential benefit comes at the cost of higher power consumption, longer cache access latencies, and increased chip area. As we move forward, the processors demand more and more number of cores which in turn the issue of providing sufficient on-chip cache capacity becomes increasingly challenging. Simply scaling cache capacity linearly with the number of cores is not practical because of power limitations and on-chip area. To resolve this issue, some prior works (e.g., [1], [2], [3], [4], [5], [6]) use various data compression schemes to achieve larger capacity without suffering all disadvantages of fabricating larger caches. The biggest obstacle in adopting cache compression in commercial processors is the decompression latency. Unlike compression, which takes place in the background, decompression is on the critical path which directly affects the system performance. Therefore, in order to improve the system performance, it is vital to achieve a fine balance between the extra capacity achieved by data compression and the extra access delay imposed by that. Studying previous compression schemes show that, compression mechanisms always sacrifice one for the other. Meaning that, sophisticated compression schemes are capable of achieving higher compression ratios (i.e., larger cache capacities) but at the cost of longer decompression latencies (i.e., slower cache accesses), and vice versa. In this work, we demonstrate that in designing a compressed cache, *data criticality* should be considered as the third design parameter, along with compression ratio and decompression latency. While typical compression schemes decide to store a cache block either in compressed or uncompressed format just based on the content of the cache block, our proposed mechanism also considers the criticality of the block. In other words, even if a cache block can be stored in a compressed format, we might decide to store it in an uncompressed or less-compressed format based on its criticality (i.e., its latency sensitivity). Based on our observations, applications exhibit different sensitivity to the access latency of different cache blocks. Such variation in latency sensitivity is partially a function of the underlying architecture where LLC is physically distributed among cores, forming a cache structure with non-uniform access latency. Considering this architecture, our proposed mechanism improves performance of the compressed cache through the following optimizations:

- Considering the fact that, (i) compression schemes always offer a trade-off between the compression ratio and decompression latency, and (ii) local and remote cache blocks exhibit different sensitivities to the access latency, we propose a hybrid mechanism to balance the extra cache capacity with the imposed decompression latency. To this end, our proposed architecture favors lower latency over higher capacity for local cache blocks by adopting a *fast compression scheme* (i.e., low-compression-ratio low-decompression-latency). For remote blocks however, we use a *strong compression scheme* (i.e., high-compression-ratio high-decompression-latency) to prioritize capacity.
- We will further discuss that, in an out-of-order execution processor, some cache blocks cause long ROB (i.e., ReOrder Buffer) stalls which directly degrades the system performance. Therefore, such blocks cannot tolerate long decompression latencies and our mechanism categorizes them as critical blocks as well. Meaning that, they can be compressed only by a fast compression scheme.

978-1-5386-1214-9/18/$31.00 ©2018 IEEE

- We will also demonstrate that, by knowledgeably adopting multiple fast compression schemes, we can improve the overall data-type/data-pattern coverage which in turn provides us a compression ratio comparable to that of strong cache compression schemes while the decompression latency is kept low.
- We will finally illustrate that, decompression process can be pipelined in a specific category of compression schemes. In such schemes, decompression process can be performed as the compressed data block traverses through the interconnection network. By performing the consecutive stages of decompression over different routers along the traversal path, we can partially overlap the decompression delay with the data traversal delay.

II. Background and Related Works

A. Baseline Platform

As shown in Figure 1, large last-level caches (LLC) in modern multi-core processors are structured as non-uniform cache architecture (NUCA) where the LLC is logically shared but physically distributed among the cores. More precisely, each cache bank is connected to one core and data movement between the banks is managed by a network of routers. In this work, we adopt a mesh topology as our network-on-chip (NOC) configuration to have an efficient low-overhead platform. In such non-uniform architecture, accesses to a local block experience a latency equal to the cache hit-latency while remote accesses experience variable latencies depending on the distance between the requesting and the target nodes.

B. Cache Compression

In the context of on-chip caches, some prior works (e.g., [1], [2], [3], [4], [5], [6]) use various data compression schemes to achieve larger cache capacity. Table I contains the characteristics of the most well-known cache compression schemes. ZCA [3] and [7] exploit zero values to compress cache blocks. FVC [5] and FPC [1] respectively use, frequently repeated data values and data patterns to encode data blocks into a compact format. BDI [4] relies on the observation that words in a cache block are close to each other, making it meaningful to code them with their difference to a base value. C-pack [2] and [8] utilize both static patterns and a dynamically updated dictionary to achieve higher coverage. SC^2 [6] leverages Huffman-based compression to achieve higher compression ratio. In general, the ultimate goal of cache compression is to achieve larger capacities while the decompression latency is kept within a reasonable range. However, compression schemes usually sacrifice one for the sake of the other.

III. Compression Implications

A. Latency versus Capacity

Data compression can potentially improve the system performance by reducing the number of off-chip memory accesses through fitting a larger portion of the working-set (WS) in the on-chip cache. However, since decompression process is on the critical path, the average data access latency will be increased which in turn can negatively outweigh the gain from

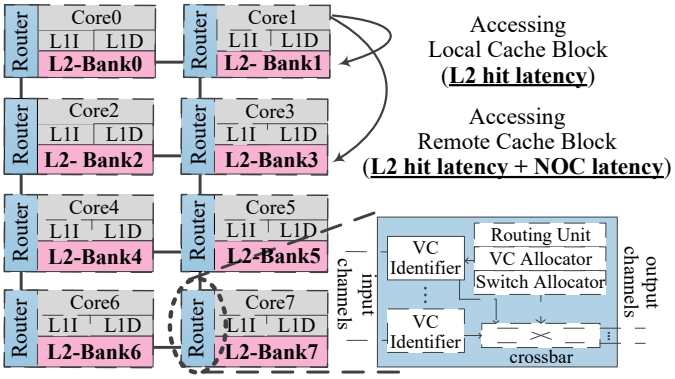


Fig. 1. Typical tiled multi-core architecture. Tiles are interconnected into a 2-D mesh. Each tile contains a core, private L1I and L1D caches, a shared L2 cache bank, and a router for data movement between nodes.

TABLE I
Cache compression techniques.

Comp. Scheme	Technical Contribution	Decomp. Latency	Comp. Ratio
ZCA [3]	Zero Values	1 Cycle	Low
FVC [5]	Frequent Values	5 Cycles	Modest
BDI [4]	Narrow Values	1 Cycle	High
FPC [1]	Frequent Patterns	5 Cycles	High
C-Pack [2]	Dynamic Dictionary	8 Cycles	High
SC^2 [6]	Statistical Compression	8/14 Cycles	High

having larger cache capacity. To further clarify this issue, Figure 2 illustrates a hypothetical compression scenario. In this figure, only WS1 fits in the baseline cache configuration where all data blocks are stored in uncompressed format. By adopting compression, WS2 can also fit in the cache along with WS1. In baseline system, WS1 elements can be accessed in a latency equal to the cache hit-latency while the latency of accessing WS2 elements is equal to the off-chip access latency. After compression however, compressed lines from WS1 and WS2 both experience a latency equal to the cache hit-latency plus the decompression latency. Therefore, in one hand, compression can degrade the performance by increasing the access latency of WS1 elements. On the other hand, it potentially improves the performance by eliminating the off-chip memory accesses for WS2 elements.

In order to have a quantitative evaluation, Figure 3 illustrates the impact of data compression on gcc and omnetpp applications under BDI [4] and FPC [1] compression schemes. Figure 3.a represents what percentage of the working-set is covered by WS1 and WS2 under different compression schemes. Figure 3.b demonstrates the performance loss caused by imposing longer access latencies to WS1 elements (1 cycle for BDI and 5 cycles for FPC as reporetd in Table I). Figure 3.c indicates the performance improvement achieved by eliminating off-chip accesses to WS2 elements. In short,

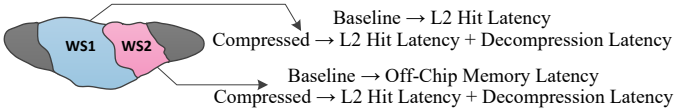


Fig. 2. Impact of compression on average data access latency. WS1: working-set that fits in baseline cache. WS2: extra portion of the working-set that fits in compressed cache.

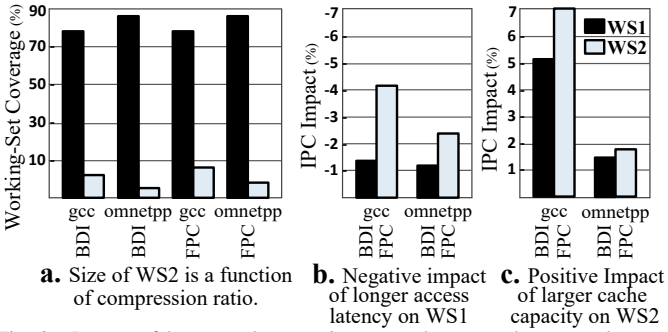
one can expect to achieve better performance only if the improvement achieved by optimizing WS2 outweighs the performance degradation caused by having slower accesses to WS1. The former is a function of compression ratio and the latter a function of decompression latency. Therefore, while a high-compression-ratio high-decompression-latency scheme can be effective for a capacity-sensitive application, it can degrade the performance for a latency-sensitive application (e.g., FPC scheme for omnetpp in Figure 3), and vice versa. Even though one can determine the ideal compression scheme for an application by performing off-line evaluations, resolving this issue at run-time is more practical and is also capable of capturing the dynamic characteristics over different phases of execution. We will demonstrate that, because applications often exhibit different sensitivity to the access latency of different cache blocks, the compression mechanism should be aware of such variations in order to achieve a fine-balanced system, where each individual cache block is handled based on its sensitivity to the decompression latency.

IV. Criticality-Aware Compression

A. Data Criticality

In general, compression schemes value different data blocks equally. Some data blocks can be compressed which subsequently occupy less space while other blocks occupy the whole cache block. In this work, on the other hand, we distinguish cache blocks based on their impact on performance. Based on our studies, computing cores are more sensitive to the access latency of some cache blocks more than others. We refer to such cache blocks as *critical* or *latency-sensitive* blocks. Considering this definition of criticality, our proposed criticality-aware compressed cache architecture distinguishes cache blocks based on their criticality and guarantees that critical blocks will be accessed in a tolerable latency. To this end, we consider two categories of critical blocks.

Local Cache Blocks: In NUCA structured caches, local cache blocks are accessed with the minimum latency which is equal to the cache hit-latency while the access latency of remote cache blocks are variable because the data has to go thought the interconnection network. Based on our studies, computing cores are more sensitive to the access latency of local cache blocks. In other words, the imposed decompression latency is less detrimental for remote cache blocks, as those blocks already experience longer access latencies compared to local blocks. For instance, in a 16-core processor structured as a 4×4 mesh topology (detailed configuration is given in table II), if we assume a uniform access traffic to different LLC banks, the access latency of a local data is 10 cycles while for a remote cache block it is about 15 cycles on average, assuming that the network is empty. For real memory-intensive applications however, the average access latency to remote cache blocks will be longer because of the traffic within the network. Therefore, we need to distinguish local and remote cache blocks because they experience widely different access latencies which subsequently affects their sensitivity to the decompression latency.



a. Size of WS2 is a function of compression ratio. **b.** Negative impact of longer access latency on WS1 **c.** Positive Impact of larger cache capacity on WS2

Fig. 3. Impact of larger cache capacity versus longer cache access latency on the system performance, in terms of IPC.

Long ROB Stalls: Most of the commercial processors in the market perform out-of-order execution to achieve maximum performance. Even though instructions are executed out-of-order, they are committed in-order. Processors typically adopt a buffer, called ReOrder Buffer (ROB) to commit the executed instructions in an in-order fashion. ROB could be stalled when the instruction at the head of ROB is not finished yet but many instructions after that are already finished and, are ready to commit. Such ROB stalls directly degrades the system performance. The memory request bind to the stalled instruction at the head of ROB is called critical load [9], [10]. Ghose et al. [9] demonstrate the importance of critical loads and accordingly proposes a memory scheduler which prioritize critical memory accesses. Kotra et al. [10] observe this issue for the cache blocks and accordingly proposes a customized NUCA platform for non-volatile LLCs. In our studies, we observe similar behaviors regarding cache blocks. Meaning that, data blocks which happen to stall ROB are more sensitive to the access latency and cannot tolerate long decompression latencies. In this work, we adopt a mechanism similar to that of [10] to detect critical cache blocks. Figure 4 depicts the structure of the criticality load predictor used in our work. On each instruction commit, the head of ROB is used to update the criticality predictor table (CPT). More precisely, PC of the head is used to index the CPT (only for load operations). If it is a hit, numLoadCount is incremented. If this load results in an ROB stall, the robBlockCount is also incremented. If the CPT does not contain an entry with the corresponding PC (miss), a new entry will be inserted into the CPT. On a lookup, CPT is accessed and based on the value of robBlockCount we determine whether the target block is critical or not[1].

[1]More information is available in [10]. Note that, in this work we do not need information such as PC viz, *LastStallTime*, *MaxStallTime*, and *TotalStallTime* because we do not have to rank the loads in terms of criticality.

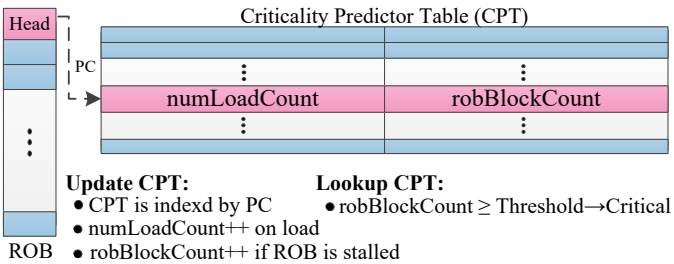


Fig. 4. Configuration of the critical load predictor logic.

B. Non-Uniform Compression

Our goal is to achieve a compressed LLC where critical data blocks are accessed with a latency comparable to that of uncompressed cache blocks while the achieved LLC capacity is comparable to that of strong compression schemes.

In Section 4.1, we defined two types of critical data blocks: (i) Local cache blocks in NUCA, and (ii) cache blocks which cause long ROB stalls. The question is how critical and non-critical data blocks should be managed to achieve large cache capacities without suffering from the imposed decompression latency. One can naively propose to only compress the non-critical cache blocks. Although this guarantees a fast access to critical blocks, we lose opportunities to achieve higher cache capacities. Additionally, based on our experimental observations, even critical cache blocks can tolerate few cycles of extra delay (i.e., 1-2 cycles) without noticeably affecting the system performance. Considering this observation, we propose a non-uniform architecture which exploits well-known compression schemes in a criticality-aware fashion. To this end, we adopt a fast compression scheme (i.e., low-compression-ratio low-decompression-latency) for critical cache blocks in order to favor lower latency over higher capacity. On the other hand, for non-critical blocks we use a strong compression scheme (i.e., high-compression-ratio high-decompression-latency) because achieving larger capacity has higher priority for this category.

Even though our hybrid architecture is compatible with any compression scheme, in this work we adopt BDI [4] and FPC [1] schemes that are easy to implement in hardware. Since BDI offers a 1-cycle decompression latency, we use that for critical data blocks. FPC on the other hand, offers a 5-cycle decompression latency and can be used for non-critical data blocks. However, in Section 6 we will demonstrate that a hybrid BDI-FPC scheme covers a wider range of data-types/data-patterns which consequently provides higher compression ratio. Therefore, for non-critical blocks, instead of just using FPC scheme, we pick the best of FPC and BDI in order to achieve larger cache capacity while for critical data blocks we always use BDI scheme. Such hybrid approaches are exploited in other works for different purposes [11], [12].

Note that, recognizing local cache blocks is straightforward as we can determine the target LLC bank based on the address of the cache request. Therefore, on a write operation, we can determine which compression scheme should be used based on the placement of the target cache block in the NUCA. For the second category of critical cache blocks however (i.e., ROB stalls), the first time a block is brought into the cache, we consider that as a non-critical block. At some point during the program execution, the criticality predictor (discussed in Section IV-A) labels that block as critical and we change its compression scheme upon the next write operation. Similar adjustment is done if a critical cache block is labeled as non-critical during the course of execution. Note also that, similar to previous studies, the tag array is doubled to be able to address twice an many cache blocks as the baseline. Besides, we keep one bit per tag entry to distinguish between BDI and FPC compressed blocks.

C. Relaxing the Decompression Latency

Data traversal in the interconnection network can be exploited as an opportunity to partially hide the decompression latency for remote data accesses. Some prior works (e.g., [13], [14]) exploit compression at the packet-level to optimize performance and power consumption of the NoC. Such techniques compress data at the network interface controller, prior to injection in the network. Unlike those techniques, we do not use compression as a knob to improve the NoC performance. Instead, we use data traversal as an opportunity to partially eliminate the decompression latency from the critical path. To do so, the decompression logic is placed in the routers which is tightly coupled with LLC banks (see Figure 1).

FPC Pipelined Decompression: Decompression process in FPC takes 5 cycles and is performed in a pipeline fashion: 1- the length of each code is calculated using the prefix tags, 2&3- the starting bit address of each word is computed, 4&5: a parallel decoder produces the uncompressed data from compressed format using the available information. In a packet-based wormhole switched network, the decompression pipeline can be designed such that decompression starts as soon as the header flit of the compressed cache block arrives. This flit contains all the necessary information for the first three stages of the decompression pipeline. When the last flit arrives, we can complete the next two stages and deliver the data to the processor. This technique effectively reduces decompression latency of FPC from 5 cycles to 2 cycles (for the cache blocks that are at least two hops away) as cache block traverses through the network. Note that, even though our proposed architecture is compatible with any compression scheme, this specific optimization is only applicable on compression schemes with pipelined decompression process.

V. Methodology

Infrastructure. We evaluate our proposed mechanism using GEM5 simulator [15]. Target system is a 16-core processor with a cache hierarchy consisting of private 32KB L1 and a shared 4MB L2. Detailed configuration is given in Table II.

Workloads. For multi-program workloads, we use the SPEC-CPU2006 benchmarks [16]. We fast-forward each workload for 2 billion instructions, warmup the caches by running 200 million instructions, and then simulate the next 200 million instructions.

TABLE II
Main characteristics of simulated system.

Processor	
CMP Configuration	ALPHA ISA, out-of-order, 16 cores @2.5GHz, 4×4 Mesh, 128 ROB entries
Memory Hierarchy	
L1 Caches	32KB/4 way, private, 1-cycle, MSHR:(4I,32D)
L2 Cache	4MB/64B/8 way, SNUCA, 10-cycle, MSHR:32
Coherency Protocol	Snooping MESI: 4×4 grid packet switched NoC; XY routing; 2 cycle per-hop latency
DRAM Memory Configuration	16GB, 4 channels, 1 DIMM/channel, 2 ranks/DIMM, 8 devices/rank, FR-FCFS, 667 MHz bus,8 Byte data bus, DDR3 1333 MHz, tRP-tRCD-CL: 15-15-15 ns, 8 DRAM banks, R-buffer hit: 36ns, R-buffer miss: 66ns

TABLE III
CHARACTERISTICS OF THE EVALUATED WORKLOADS FOR LAST-LEVEL CACHE.

Workload	MPKI	BDI	FPC	Hybrid_Comp	Workload	MPKI	BDI	FPC	Hybrid_Comp
MP1: namd, xalancbmk, zeusmp, gcc	2.3	5.2	2.2	7.1	MP8: bzip, gcc, xalancbmk, zeusmp	8.9	2.5	1.7	2.8
MP2: gcc, GemsFDTD, gromacs, h264ref	0.7	1.5	1.9	2.2	MP9: xalancbmk, xalancbmk, gcc, gcc	2.65	1.3	1.7	1.8
MP3: soplex, sphinx3, tonto, xalancbmk	0.5	1.4	1.7	1.8	MP10: deal, soplex, namd, bzip2	4.15	1.6	1.5	2.0
MP4: namd, tonto, cactusADM, dealII	1.5	2.0	1.7	2.6	MP11: sphinx3, tonto, zeusmp, gcc	10.5	12.8	2.2	16
MP5: xalancbmk, gcc, namd, omnetpp	1.0	1.3	1.7	1.8	MP12: bzip, zeusmp, xalancbmk, dealII	10.8	2.4	1.7	2.9
MP6: xalancbmk, cactusADM, dealII, gcc	2.7	1.8	1.7	2.4	MP13: deal, bzip2, zeusmp, xalancbmk	11.0	2.3	1.8	3.0
MP7: omnetpp, omnetpp, gcc, gcc	1.6	1.3	1.7	1.9	MP14: bzip2, cactusADM, dealII, gcc	8.85	1.8	1.6	2.2

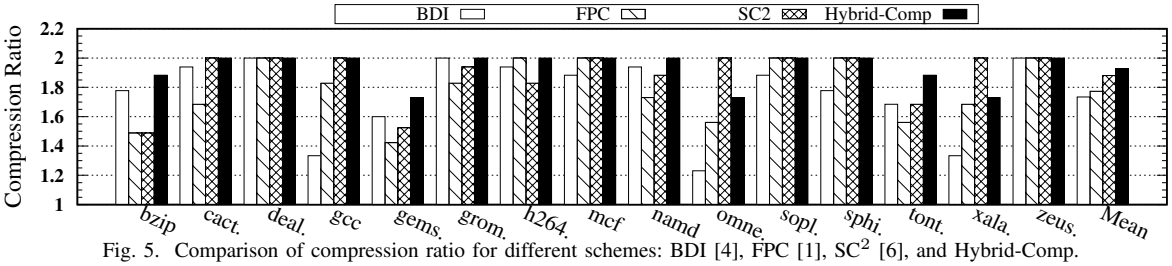


Fig. 5. Comparison of compression ratio for different schemes: BDI [4], FPC [1], SC^2 [6], and Hybrid-Comp.

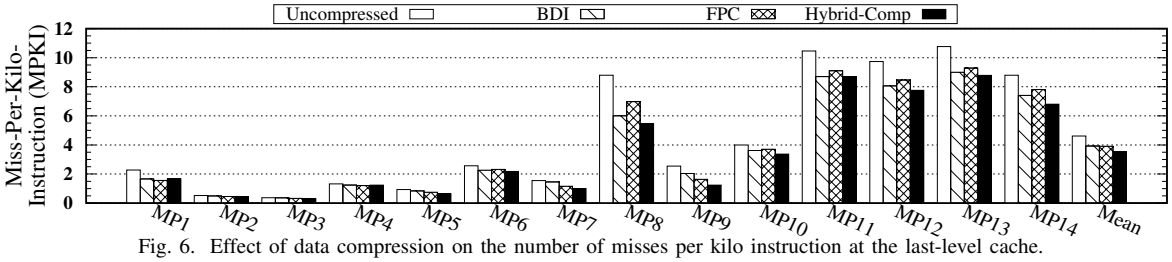


Fig. 6. Effect of data compression on the number of misses per kilo instruction at the last-level cache.

VI. EVALUATION

A. Compression Ratio

Figure 5 demonstrates the compression ratio of different compression schemes over SPEC2006 [16] applications. As discussed in Section 2, BDI uses narrow data values to compress a cache block. FPC on the other hand, exploits frequent data patterns. Employing these two schemes in a unified architecture can cover a wider range of data types which considerably improves the average compression ratio. Our proposed architecture, uses these two schemes in a hybrid criticality-aware fashion and achieves a compression ratio comparable to that of SC^2 [6]. Note that, SC^2 uses Huffman-based statistical compression and has a decompression latency of 8/14 cycles (depending on the position of critical word in the cache block), while our proposed architecture achieves a 1-5 cycles decompression latency (depending on the type of compression scheme and position of data in NUCA). Therefore, our proposed architecture outperforms SC^2 in terms of both the compression ratio, owing to the data type coverage achieved by using multiple compression schemes, and the decompression latency. Note that, in the compressed cache, the tag array is doubled to be able to address twice as many cache lines as the baseline. Therefore, the maximum compression ratio reported in Figure 5 is limited to 2. SC^2 can outperform our approach (in terms of compression ratio) for some applications if we use larger tag caches.

Note that, Arelakis et al. [12] also propose a hybrid compression mechanism. However, they focus on the content of each cache block (i.e., run-time data-type prediction) rather than its criticality, and is orthogonal to our work.

B. Misses-Per-Kilo-Instructions (MPKI)

MPKI can be used as a metric to analyze the caching efficiency in cache-sensitive applications. As can be seen in Figure 6, MPKI is reduced in compressed caches, thanks to the extra cache capacity achieved by compression. Table III reports the compression ratios of BDI, FPC, and our hybrid architecture for the studied workloads. Comparing the reported compression ratios in Table III with MPKI in Figure 6 demonstrates the direct impact of compression ratio on MPKI. Our proposed architecture reduces MPKI by 25% on average compared to the baseline cache configuration while it outperforms BDI and FPC schemes by about 10% on average.

C. Average Data Access Latency

Figure 7 reports average data access latency in LLC normalized to the baseline cache with no compression. Since we have doubled the size of tag array in the compressed cache, it can at most pack twice as many cache lines as the baseline cache. Therefore, as shown in Figure 7, the average data access latency for a double size LLC (i.e., 8MB) indicates the maximum improvement that can be achieved by compression. In a compressed cache, decompression latency determines the latency of hit accesses while compression ratio affects the hit-rate. Comparing average data access latency of FPC and BDI demonstrates the importance of decompression latency. Even though, for about half of the applications FPC provides better compression ratio compared to BDI, for all the workloads reported in Figure 7, FPC achieves longer access latencies (in some cases even worse than the baseline) because of its long decompression process. Our proposed architecture on the

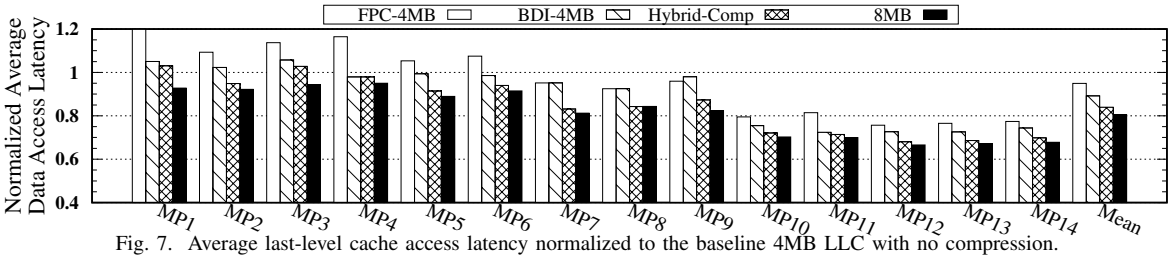


Fig. 7. Average last-level cache access latency normalized to the baseline 4MB LLC with no compression.

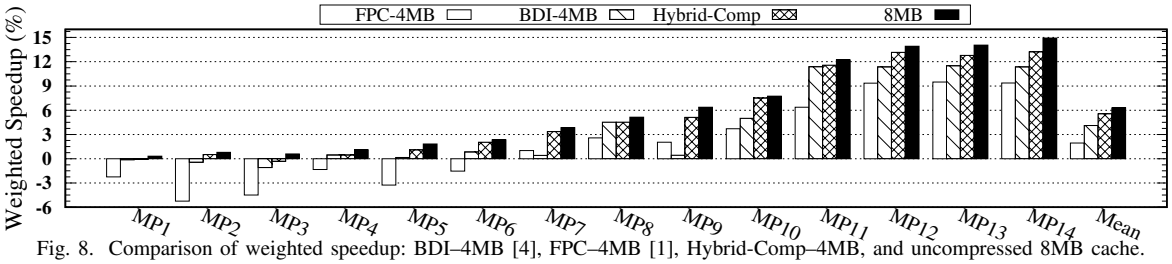


Fig. 8. Comparison of weighted speedup: BDI–4MB [4], FPC–4MB [1], Hybrid-Comp–4MB, and uncompressed 8MB cache.

other hand, outperforms FPC and BDI in terms of compression ratio, and has a decompression latency comparable to that of BDI. Therefore, as can be seen in Figure 7, our proposed architecture outperforms both BDI and FPC, and achieves an average access latency close to that of a double size LLC.

D. Performance

Figure 8 demonstrates the impact of different compression techniques on the system performance in terms of weighted speedup. Latency-sensitive workloads (i.e., workloads with low MPKI shown in Figures 6) do not gain from the extra last-level cache capacity achieved by compression; however, the imposed decompression latency can degrade their performance. For instance, comparing Figures 6 and 8 illustrates that, workloads with low MPKI (e.g., MP1 to MP5) incur performance loss under FPC cache compression scheme. Our proposed mechanism however, do not experience any performance loss for such latency-sensitive workloads as it fundamentally prioritize latency over capacity. On the other hand, capacity-sensitive workloads (i.e., workloads with high MPKI reported in Figures 6) considerably gain from the extra last-level cache capacity achieved by data compression (e.g., MP10 to MP14 in Figure 8). Overall, our proposed mechanism outperforms both FPC and BDI, thanks to its lower average data access latency, and achieves a performance improvement (i.e., 5.3% on average) comparable to that of an 8MB uncompressed LLC (i.e., 6% on average).

Note that, simply adopting a hybrid FPC-BDI scheme makes an average improvement of only 1.4%. By considering critical and non-critical data blocks we can achieve an improvement of 4.1% on average. Finally by using the decompression relaxing technique we reach an average improvement of 5.3%.

VII. Conclusion

In this paper, we propose data criticality as a parameter that should be considered in designing compressed caches, along with compression ratio and decompression latency. Based on our studies on a 16-core processor with 4MB last-level cache, the proposed criticality-aware architecture improves the system performance comparable to that of with an 8MB LLC.

Acknowledgment

This work is supported in part by NSF grants 1526750, 1302557, 1213052, 1439021, 1626251, 1409095, 1629915, 1629129, and 1302225 and a grant from Intel.

References

[1] A. R. Alameldeen *et al.*, "Adaptive cache compression for high-performance processors," *SIGARCH Comput. Archit. News*, Mar. 2004. [Online]. Available: http://doi.acm.org/10.1145/1028176.1006719

[2] X. Chen *et al.*, "C-pack: A high-performance microprocessor cache compression algorithm," *IEEE Trans. VLSI*, 2010. [Online]. Available: http://dx.doi.org/10.1109/TVLSI.2009.2020989

[3] J. Dusser *et al.*, "Zero-content augmented caches," in *ICS*, 2009. [Online]. Available: http://doi.acm.org/10.1145/1542275.1542288

[4] G. Pekhimenko *et al.*, "Base-delta-immediate compression: Practical data compression for on-chip caches," ser. PACT '12. [Online]. Available: http://doi.acm.org/10.1145/2370816.2370870

[5] J. Yang *et al.*, "Frequent value compression in data caches," in *MICRO*, 2000. [Online]. Available: http://doi.acm.org/10.1145/360128.360154

[6] A. Arelakis and P. Stenstrom, "Sc2: A statistical compression cache scheme," ser. ISCA '14. [Online]. Available: http://dl.acm.org/citation.cfm?id=2665671.2665696

[7] M. Arjomand *et al.*, "A morphable phase change memory architecture considering frequent zero values," in *ICCD*, Oct 2011, pp. 373–380.

[8] M. Arjomand, A. Jadidi, M. T. Kandemir, and C. R. Das, "Leveraging value locality for efficient design of a hybrid cache in multicore processors," in *ICCAD*, Nov 2017, pp. 1–8.

[9] S. Ghose *et al.*, "Improving memory scheduling via processor-side load criticality information," in *ISCA*, 2013. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485930

[10] J. B. Kotra *et al.*, "Re-nuca: A practical nuca architecture for reram based last-level caches," in *IPDPS*, 2016.

[11] A. Jadidi *et al.*, "Exploring the potential for collaborative data compression and hard-error tolerance in pcm memories," in *DSN*, June 2017, pp. 85–96.

[12] A. Arelakis *et al.*, "Hycomp: A hybrid cache compression method for selection of data-type-specific compression methods," ser. MICRO. ACM, 2015, pp. 38–49. [Online]. Available: http://doi.acm.org/1d0.1145/2830772.2830823

[13] Y. Jin *et al.*, "Adaptive data compression for high-performance low-power on-chip networks," ser. MICRO 41. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2008.4771804

[14] P. Zhou *et al.*, "Frequent value compression in packet-based noc architectures," ser. ASP-DAC '09. [Online]. Available: http://dl.acm.org/citation.cfm?id=1509633.1509640

[15] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, 2011. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718

[16] C. D. Spradling, "SPEC CPU2006 benchmark tools," *SIGARCH CAN*, vol. 35, no. 1, pp. 130–134, Mar. 2007.