Dynamic Cache Pre-Decompression

Hassan Farooq
University of Illinois
hfaroo9@illinois.edu

Spenser Fong
University of Illinois
sfong5@illinois.edu

Timothy Vitkin
University of Illinois
tvitkin2@illinois.edu

I. Background and Related Work

Accessing data in main memory can be very costly as it can take a significant amount of clock cycles, especially in comparison to data access from caches. Increasing cache size is not always a viable alternative in order to move more data closer to the main core simply due to limited die area, additional cost, additional heat, and additional power necessary. Among various methods, cache compression has been popular previously proposed technique to artificially increase the amount of cache available without substantially increasing the physical size of the caches. Even with the additional computation overhead of compression and decompression, cache compression can be significantly faster than main memory access and thus beneficial for performance. Due to the overhead involved, cache compression techniques tend to mainly be utilized in last-level cache (LLC). The issue here is that while less space is necessary for the same amount of data, more time is needed before the data is in a state in which it can be used (ie. uncompressed data) in comparison to a traditional cache. Mitigating this overhead could allow for decreased wait time for memory and improved performance and provide justification in using similar techniques in lower level caches, thus further artificially increasing the amount of space available in the caches.

Compressing data in caches close to the main core is tricky due to the overhead and additional latency that occurs with the decompression necessary before data can be used [1]. Due to this, most recent work investigates compression techniques in the LLC of the processor in order to increase effective cache capacity while keeping latency and physical cache size low [2]. This is especially important when using an aggressive, stride-based prefetcher that pollutes the cache, causing degradation in performance. In such cases, compression provides improved performance of up to 50% [3].

Previous work on using cache compression includes partially compressed caches, meaning that the cache has a partition for compressed data and a separate partition for uncompressed data [4], [5]. Some compression algorithms have been explored to increase compression speed, account for different data types, and reduce energy consumption [6]–[8]. For instance, by using chunks of four cache lines ("superblocks") Sardashti et al. were able to reduce tag overhead in compressed caches, resulting in improved effective cache capacity without the need of significant metadata, backward pointers, or the complexity of skewed associativity [9].

Compressed LLCs that use data criticality as a consideration during compression have also been proposed, leading to 4MB compressed LLCs having performance comparable to that of an 8MB uncompressed LLC [10]. Combining compression with various replacement policies has also been explored to try and improve performance. It was found that advanced replacement policies interact poorly with compression, resulting in no noticeable improvements between compressed and uncompressed caches, but with an opportunistic cache compression mechanism, improvements of up to approximately 9% could be seen [11]. Techniques other than compression have also been tested to try and increase effective cache capacity. For instance, Huang et al. proposed a critical-words-only cache in which only words that are generally accessed before others are kept in cache resulting in a 256kB L2 cache performing just as well as a 512kB conventional L2 cache on average [12].

Due to the delay of decompression, certain techniques have been explored in order to hide the latency penalty incurred during this step [13]. Alameldeen et. al proposed a technique where an LRU and compressed data size determined whether or not data should be stored compressed or decompressed, providing an improvement of 17% for memory-intensive benchmarks. They also note that while on a typical compressed cache, a memory-intensive benchmark with a low cache miss rate incurred a performance penalty of 18%, their adaptive-compression cache only caused a 0.4% performance penalty [13]. Rea et al. proposed cache compression in addition to data prefetching in order to offset some of the latency caused by decompression specifically in L1 caches [14]. It was found that a base-delta-immediate compression combined with stride/last outcome prefetching allowed for a 1.7% average speedup compared to simply using compression without prefetching [14]. Lee et al. suggested using selective compression, parallel decompression, and the use of decompression buffers to reduce decompression overhead resulting in a 35-53% reduction in data traffic and a 20% reduction in average memory access time [15].

II. Research Problem Statement

One way to exploit the benefits of larger caches without increasing the area the cache takes up is cache compression, which compacts data stored in the cache. Cache compression can lead to higher data bandwidth and reduced overall energy consumption, at the cost of hit latency. Reducing overhead between memory requests for compressed data in caches and the data being returned to the main core can be beneficial to

performance and allow for similar compression techniques to be applied in additional levels of cache. This could further increase effective cache capacity.

We propose a dynamic cache decompression technique, where cacheline entries are dynamically reallocated to either the compressed or uncompressed partitions of the cache in order to hide decompression latency. Our technique will use in-cache computing. Similar to prefetching, this would occur without the need of the main core and without its knowledge. In essence, our cache will appear to be the same as a traditional cache from the perspective of the main core.

III. Proposed Ideas and Solutions

The high level idea is to smartly choose data to decompress before it's actually needed. The system will utilize runahead execution to speculatively decompress data. Another approach is to look into other heuristics for tagging data to be dynamically compressed/decompressed.

We could additionally attempt to combine our cache compression/decompression techniques with other methods to increase effective cache capacity. This could include extending the critical-words-only cache method proposed by Huang et al. and adding compression [12]. A combination of various techniques could further make better use of valuable cache space at the cost of complex logic elements and potentially large delays when certain values are incorrectly predicted to need decompression or are compressed too early.

IV. Implementation Approaches

The gem5 simulator by default includes several cache compression algorithms, each of which are implemented using C++ and organized using Python. The simulator also includes a framework to change the indexing and replacement policies for a cache. Using these options, we plan on adding classes to the Python configuration file to support our compression/decompression algorithms, and then implement them in C++.

V. Execution Plan

We plan to use gem5 to simulate CPU and cache compression techniques. We will extend the preexisting gem5 compression code to include prediction and dynamic compression/decompression.

A. Weekly Milestones

- Week 1 (Oct 31 - Nov 7) - Setting up a basic compression algorithm in gem5, which may include extending parts of the simulator. Metrics and workloads will also be decided during this period.
- Week 2 (Nov 7 - Nov 14) - Running test workloads on preexisting compression/decompression techniques and implementation of our own decompression technique
- Week 3 (Nov 14 - Nov 21) - Initial run of tests on our technique and comparison between our and preexisting techniques
- Week 4 (Nov 21 - Nov 28) - Fall break
- Week 5 (Nov 28 - Dec 5) - Implementation of additional decompression techniques and comparisons as time allows
- Week 6 (Dec 5 - Dec 12) - Creating final presentation and finishing final project report

VI. Estimated Output

We plan on analyzing similar statistics to what we looked at in MP1. This includes program execution time, instructions per cycle (IPC), memory bandwidth, cache hit rate, and power statistics across the x86 and RISC-V ISAs and a variety of different processor configurations. We will also test using our compression/decompression techniques on multiple levels of caches. We expect slight increases in power usage ($\sim$10% increase) in comparison to other compression techniques due to our more complex logic and expect slightly increased memory cache hit rate ($\sim$10% increase) and decreased execution time ($\sim$5% decrease). The percentages are based on a similar paper by Rea et al. on cache compression/decompression techniques and results they observed [14].

References

[1] A. R. Alameldeen and R. Agarwal, "Opportunistic compression for direct-mapped dram caches," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 129–136. [Online]. Available: https://doi.org/10.1145/3240302.3240429

[2] D. Chen, E. Peserico, and L. Rudolph, "A dynamically partitionable compressed cache," 2003. [Online]. Available: https://dspace.mit.edu/handle/1721.1/3677

[3] A. R. Alameldeen and D. A. Wood, "Interactions between compression and prefetching in chip multiprocessors," in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, 2007, pp. 228–239.

[4] S. Singh, J.-M. Cheng, B. C. Beardsley, D. P. Leabo, F. L. Wade, M. T. Benhase, and M. E. Goldfeder, "Data caching with a partially compressed cache," Patent, Nov, 2001. [Online]. Available: https://patents.google.com/patent/US6324621B2

[5] D. R. Carvalho and A. Seznec, "Understanding cache compression," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 3, jun 2021. [Online]. Available: https://doi.org/10.1145/3457207

[6] 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*, vol. 18, no. 8, pp. 1196–1208, 2010.

[7] L. Villa, M. Zhang, and K. Asanović, "Dynamic zero compression for cache energy reduction," in *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, ser. MICRO 33. New York, NY, USA: Association for Computing Machinery, 2000, p. 214–220. [Online]. Available: https://doi.org/10.1145/360128.360150

[8] A. Arelakis, F. Dahlgren, and P. Stenstrom, "Hycomp: A hybrid cache compression method for selection of data-type-specific compression methods," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO-48. New York, NY, USA: Association for Computing Machinery, 2015, p. 38–49. [Online]. Available: https://doi.org/10.1145/2830772.2830823

[9] S. Sardashti, A. Seznec, and D. A. Wood, "Yet another compressed cache: A low-cost yet effective compressed cache," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 3, sep 2016. [Online]. Available: https://doi.org/10.1145/2976740

[10] A. Jadidi, M. Arjomand, M. T. Kandemir, and C. R. Das, "Hybrid-comp: A criticality-aware compressed last-level cache," in *2018 19th International Symposium on Quality Electronic Design (ISQED)*, 2018, pp. 25–30.

[11] J. Gaur, A. R. Alameldeen, and S. Subramoney, "Base-victim compression: An opportunistic cache compression architecture," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. IEEE Press, 2016, p. 317–328. [Online]. Available: https://doi.org/10.1109/ISCA.2016.36

[12] C.-C. Huang and V. Nagarajan, "Increasing cache capacity via critical-words-only cache," in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, 2014, pp. 125–132.

[13] A. R. Alameldeen and D. A. Wood, "Adaptive cache compression for high-performance processors," *SIGARCH Comput. Archit. News*, vol. 32, no. 2, p. 212, mar 2004. [Online]. Available: https://doi.org/10.1145/1028176.1006719

[14] S. Rea and E. Atoofian, "Mitigating critical path decompression latency in compressed l1 data caches via prefetching," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018, pp. 694–701.

[15] J.-S. Lee, W.-K. Hong, and S.-D. Kim, "An on-chip cache compression technique to reduce decompression overhead and design complexity," *Journal of Systems Architecture*, vol. 46, no. 15, pp. 1365–1382, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762100000308