NATURALISTA CAMPANO
ISSN: 1827-7160
Volume 28 Issue 1, 2024

C-Pack: Enhancing Microprocessor Performance through Cache Compression

Mrs. Siva pavani Veeranalla[1*], Dr. Kunchala Little Flower[2], Dr. Santhosh Boddupalli[3], Dr. Pandi Chiranjeevi[4]

[1*]Assistant Professor, Computer Science and Engineering, Malla Reddy Engineering College (Autonomous), Maisammaguda, Secundeabad.
[2]Assistant Professor, Department of CSE, Malla Reddy College of Engineering and Technology, Maisammaguda, Secunderabad.
[3]Assistant Professor, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, KL (Deemed to be University),Hyderabad - 500075, Telangana, India.
[4]Associate Professor & HOD, Dept CSE(Data Science) ACE Engineering College, Hyderabad, India.

Email: [1]pavaniveeranala3@gmail.com, [2]littleflowerk@gmail.com, [3]dr.santhoshb.klu@gmail.com, [4]chiruanurag@gmail.com

Abstract: Researchers in computer systems and micro-architecture have advocated for the integration of hardware data compression units into microprocessors' memory hierarchies to enhance performance, energy efficiency, and functionality. Nevertheless, prior research, particularly in cache compression, has often relied on unverified assumptions regarding the performance, power consumption, and area overheads of proposed compression algorithms and hardware implementations. In this study, a novel lossless compression algorithm tailored for rapid online data compression, specifically targeting cache compression, is introduced. This algorithm incorporates innovative features, such as the consolidation of compressed lines into single cache lines and the ability to compress multiple words in parallel using a single dictionary, all without compromising compression ratio. Additionally, the proposed algorithm is translated into a register transfer level hardware design, allowing for the estimation of performance, power consumption, and area requirements

Keywords: Cache Compression, Hardware Implementation, Pair Matching, Parallel Compression.

1. Introduction

This study addresses the pressing issue of managing off-chip communication in computer systems to ensure optimal performance and energy efficiency. With microprocessor speeds outpacing off-chip memory latency, a notable "wall" forms between processor and memory access. The shift towards chip-level multiprocessors (CMPs) exacerbates this challenge, as increased processors lead to more memory accesses, while the performance of the processor-memory bus struggles to keep pace. Implementing techniques to reduce off-chip communication without compromising performance presents a promising solution. Cache compression emerges as a pivotal strategy, compressing data within last-level on-chip caches like L2 caches to yield larger usable cache capacities. Previous research [1] highlights significant performance enhancements with cache compression, showing improvements of up to 17% for memory-intensive commercial workloads and even up to 22.5% for memory-intensive scientific workloads [2]. Furthermore, combining cache compression with perfecting techniques has shown throughput improvements ranging from 10% to 51% in CMP scenarios [3]. However, prior studies have not adequately validated whether the proposed compression/decompression hardware is optimized for cache compression, considering performance, area, and power consumption requirements. Such validation is essential for accurately assessing the performance impact of adopting cache compression. Cache compression introduces several challenges. Firstly, decompression and compression operations must be swift to avoid a notable increase in cache hit latency, which could offset the benefits of reduced cache miss rates. Secondly, hardware implementation should occupy minimal area relative to the reduction in cache size and should not substantially increase total chip power consumption [4]. Thirdly, the compression algorithm must effectively compress small blocks, such as 64-byte cache lines, while maintaining a favorable compression ratio. Traditional compression metrics like block compression ratio are insufficient; instead, evaluating the effective system-wide compression ratio is essential [5].

Furthermore, this paper introduces novel quality metrics for evaluating cache compression algorithms. Lastly, any cache compression technique must not significantly increase power consumption, ruling out the use of high-overhead compression algorithms. A faster and lower-overhead approach is crucial to meeting these rigorous requirements. The remaining parts of the article have been arranged as follows. In part 2 we introduce related work of the paper, in section 3 we present our proposed approach, in section 4 we present valuation of the proposed approach and finally we conclude our paper in section 5.

2. Related Work

Decrypting compressed data is often faster than decrypting uncompressed data due to the reduced amount of data that needs processing [6]. Additionally, during data transmission over networks, compression can decrease the required bandwidth, leading to faster speeds and lower network costs. Furthermore, compressed data takes up less memory space, which can enhance system performance by reducing memory usage and the occurrence of cache misses [7]. Nevertheless, it's important to acknowledge potential trade-offs. Compression and decompression tasks utilize CPU resources, which might affect overall system performance, particularly in situations where the CPU is heavily utilized. Additionally, highly compressed data may demand more computational resources for decompression, potentially causing latency issues, especially if decompression happens in critical system components. In summary, while data compression provides various advantages such as decreased storage requirements, optimized bandwidth usage, and enhanced system performance, it's essential to assess its impact on latency and resource consumption to determine the most appropriate approach for a specific use case Improving data security can be accomplished by employing various data transmission methods[8]. Ensuring transparency in both compression and decompression processes to potential threats adds an extra layer of security to the system. Additionally, utilizing compressed data formats has the potential to speed up input and output operations on computer devices. Implementing data compression for storing large database files can lead to significant cost reductions across different applications. By harnessing the advantages of data compression, the development of multimedia and video applications becomes more feasible at a lower cost, thereby increasing accessibility to such innovations. The basic block diagram of the Data Compression Model, illustrated in Figure 1, delineates the essential stages of data compression. These stages encompass data redundancy reduction, entropy reduction [9], and entropy encoding. Data redundancy occurs when a field within a database management system is duplicated across multiple tables [10]. This redundancy can introduce inconsistencies and compromises in data integrity, underscoring the importance of mitigating redundancy in relational database design. Database normalization methods are instrumental in addressing redundancy and optimizing storage efficiency. Sameer et al [11] proposed a model to improve the performance of cache, in this method the authors used rectangular matrix multiplication, to get the desired output, however the designed method is too complex and improves delay. K.Kalpana et al [12] proposed a technique to improve security to FPGA processor, in which the authors used compression techniques.

3. Proposed Approach

Figure 1 represents the basic building diagram of data compression model, in which The CPU communicates with its L1 cache, using compression to improve data storage efficiency within the cache's limited space. The L2 cache, positioned between the L1 cache and main memory, also employs compression techniques to maximize its storage capacity. Main memory acts as the central storage for the system, storing data in its original, uncompressed format. In this version, the language is refined for clarity and conciseness while maintaining the emphasis on the benefits of cache compression for microprocessor performance.

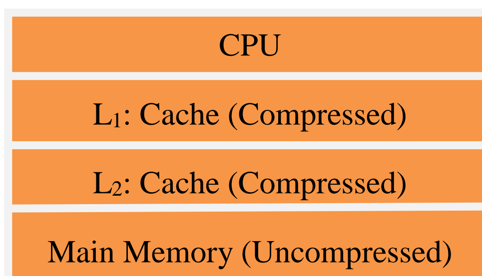

Figure 1: Data Compression Model

3.1. Optimizing Cache Compression Architecture: Strategies for Enhanced Performance

In this section, we delve into the architecture of a CMP (Chip Multiprocessor) system, focusing on the integration of cache compression techniques. Our emphasis lies particularly on private on-chip L2 caches, known for their consistent design attributes, especially as the number of processor cores scales up. Additionally, we explore the incorporation of data perfecting techniques to further boost system performance. The schematic overview provided in Fig. 1 illustrates a CMP system comprising n processor cores, each equipped with private L1 and L2 caches. Within the L2 cache, we distinguish between two regions: an uncompressed segment (referred to as "L2" in the figure) and a compressed region (labeled as "L2C" in the diagram). The sizes of these regions can be either statically configured or dynamically adjusted based on the specific requirements of each processor. In scenarios where capacity constraints arise, the entire L2 cache may be compressed, while in others, it may remain uncompressed to minimize access latency.

This architecture entails a three-level cache hierarchy comprising the L1 cache, the uncompressed L2 region, and the compressed L2 region. Direct communication occurs between the L1 cache and the uncompressed L2 region. Data exchange between the uncompressed and compressed regions is facilitated through a compressor and decompressor mechanism. This setup allows for the compression of uncompressed cache lines within the compressor, followed by their placement into the compressed region, and vice versa. The compressed L2 region serves as a virtual layer within the memory hierarchy, offering a larger size but entailing higher access latency compared to the uncompressed L2 region. It's noteworthy that the techniques discussed here can be seamlessly applied to systems with shared L2 caches without necessitating architectural modifications. The primary difference lies in the fact that both cache regions would contain lines from various processors, rather than being exclusive to a single processor, as observed in private L2 caches.

3.2. Refining the C-PACK Compression Algorithm

In this section, we present an overview of the C-Pack compression algorithm. Initially, we provide a succinct description of the algorithm along with its essential features, highlighting elements that favor efficient hardware implementation. We acknowledge that these considerations may differ from those for software implementation. Furthermore, we explore the design trade-offs associated with C-Pack and evaluate its efficacy within compressed-cache architecture.

3.2.1. A. Design Constraints and Challenges

At the outset, we underscore several design constraints and challenges inherent to the cache compression problem.

1. Cache compression demands hardware capable of rapidly compressing or decompressing a word within a few CPU clock cycles. This precludes software implementations and profoundly influences the design of compression algorithms.
2. Cache compression algorithms must maintain losslessness to ensure the accuracy of microprocessor operations
3. Compared to other compression applications like file and main memory compression, cache compression typically deals with smaller block sizes. Consequently, achieving a low compression ratio poses a significant challenge.
4. Managing the locations of cache lines post-compression introduces complexity that can impact feasibility. Allowing arbitrary, bit-aligned locations would overly complicate cache design and render it impractical. Therefore, a scheme that facilitates fitting a pair of compressed lines within an uncompressed line proves advantageous.

3.2.2. Overview of the C-Pack Algorithm

C-Pack, short for Cache Packer, represents a meticulously engineered lossless compression algorithm m specifically tailored for high-performance hardware-based on-chip cache compression. It demonstrates exceptional proficiency in attaining commendable compression ratios, especially when handling data commonly found in microprocessor low-level on-chip caches like L2 caches. The algorithm's design is significantly influenced by earlier research on pattern-based partial dictionary match compression. However, it's important to highlight that this prior research primarily centered on software-based main memory compression and didn't address the nuances of hardware implementation.

3.2.3. Pack achieves compression through two distinct methods:

NATURALISTA CAMPANO
ISSN: 1827-7160
Volume 28 Issue 1, 2024

- ✓ It employs compact encodings for frequently encountered data words, predetermined for static efficiency.
- ✓ The system utilizes a dynamic dictionary that adapts to commonly found words, supporting both full and partial word matching. These encoding strategies, along with the corresponding patterns and their frequencies in cache trace data, are outlined in Table I. Here, 'z' denotes a zero byte, 'm' indicates a byte matched against a dictionary entry, and 'x' represents an unmatched byte. In the 'Output' column, 'B' signifies a byte, and 'b' represents a bit.

The compression and decompression processes of C-Pack are illustrated in Figure 2, showcasing an example with two words per cycle. Nonetheless, the algorithm can be flexibly extended to accommodate scenarios with varying word counts per cycle. During processing, each word undergoes comparison with predefined patterns such as "zzzz" and "zzzx". If a match is detected, the compression output combines the relevant code with any unmatched bytes, as specified in Table I. Alternatively, if no match is found, the compressor evaluates all dictionary entries to identify the one with the highest match count.

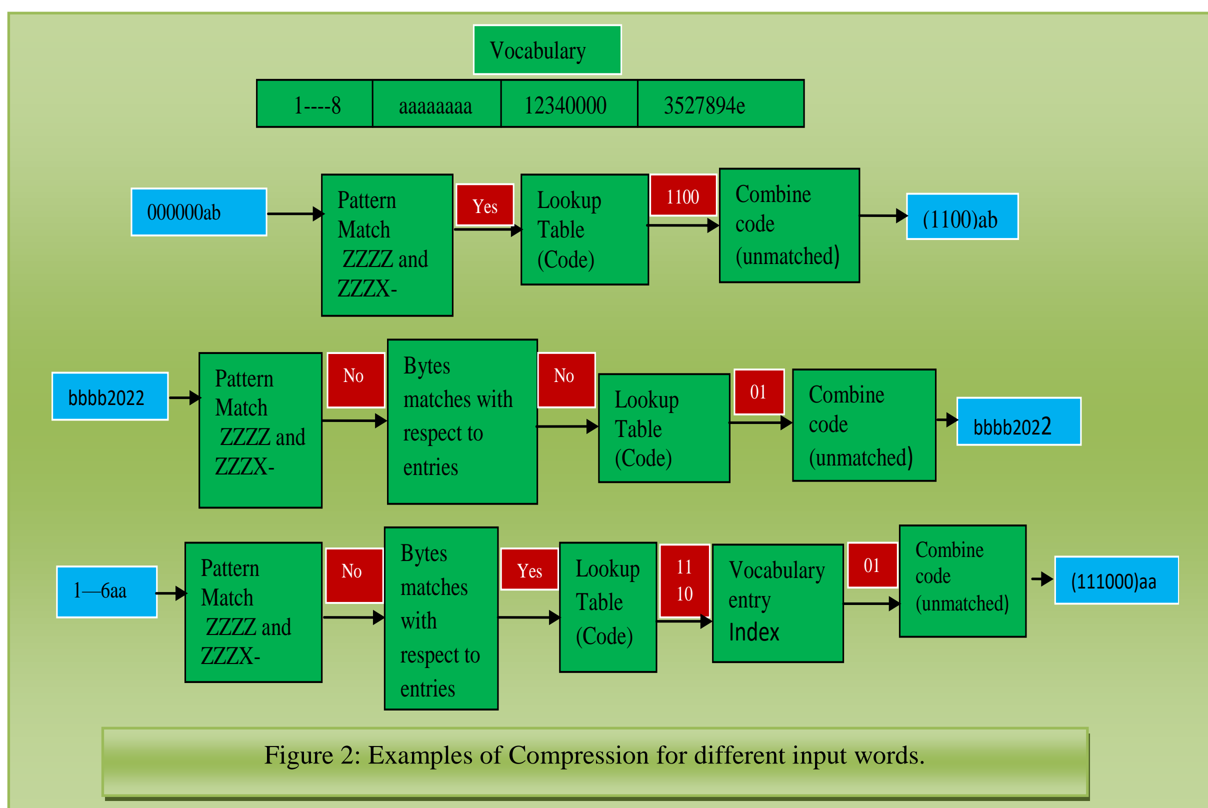

Figure 2: Examples of Compression for different input words.

Table 1: Encoding pattern for C- Pack

Code	Pattern	O/P	Length	Frequency (%)
00	zzzz	(00)	2	39.7
01	xxxx	(01)bbbb	34	32.1
10	mmmm	(10)BBBB	6	7.6
1100	mmxx	(1100)BBBBbb	24	6.1
1101	zzzx	(1101)B	12	7.3
1110	mmmx	(1110)BBBBb	16	7.2

The compression process combines the code, dictionary entry index, and any unmatched bytes to produce the final output. If words fail to match any patterns, they are included in the dictionary. While Fig. 2 illustrates compression results with a 4-word dictionary, our implementation employs a 64 B dictionary, updated after each word insertion. During decompression, the de-compressor reads compressed words, analyzes their patterns using extracted codes, and compares them against predefined patterns in Table I. If a code indicates a pattern match, the original word is reconstructed by incorporating zeroes and any unmatched bytes. Alternatively, if a code signifies a dictionary match, the decompression output combines bytes from the input word with bytes from

vocabulary entries. C-Pack's algorithm is optimized for hardware implementation, leveraging parallel comparison of input words with multiple patterns and dictionary entries. This parallel approach enables efficient execution with favorable compression ratios in hardware implementations, albeit with potential limitations in software implementations due to serialized operations. Nonetheless, the parallel design of C-Pack supports simultaneous pattern matching, dictionary matching, and processing of multiple words, with careful consideration of design parameters such as dictionary replacement policy and coding scheme to manage hardware complexity while maintaining effective system-wide compression ratios.

In our proposed implementation of C-Pack, two words are processed in parallel per cycle. Achieving this by ensuring accurate dictionary matches for both words presents a challenge. When compressing two similar words not recently encountered by the compression algorithm, the appropriate dictionary content for the second word depends on whether the first word matched a static pattern. If it did, the first word won't be in the dictionary; otherwise, it will be, aiding in encoding the second word. Therefore, the second word should be compared with the first word and all dictionary entries except the first in parallel. This approach improves compression ratio compared to omitting comparison with the first word, enabling parallel compression of two words without sacrificing compression efficiency

Valuation

- **C-Pack Synthesis Results:**

Our design underwent synthesis using Synopsys Design Compiler across various technology libraries. In a 65 nm technology, our hardware design achieves a throughput of 80 Gb/s for compression and 76.8 Gb/s for decompression, with manageable area and power consumption overheads. Specifically, the total power consumption at 1 GHz stands at 48.82 mW, representing only 7% of the total power consumption of a 512 KB cache with a 64 B block size at the same frequency.

- **Comparison of Compression Ratio:**

We compared C-Pack with several other hardware compression designs suitable for cache compression, including X-Match, FPC, and MXT. Testing on various cache data traces revealed that X-Match exhibits the best raw compression ratios, while MXT shows the poorest performance on average due to its limited dictionary size. However, in terms of effective system-wide compression ratios, X-Match leads with C-Pack closely following. On average, C-Pack surpasses FPC and MXT by 6.78% and 10.3%, respectively.

- **Comparison of Hardware Performance:**

This section compares C-Pack's decompression latency, peak frequency, and area with those of MXT, X-Match, and FPC. Notably, despite MXT's implementation in a memory controller chip operating at 133 MHz, its decompression rate lags behind C-Pack's efficiency. X-Match boasts a maximum frequency of 50 MHz and a throughput of 200 MB/s, but C-Pack's implementation achieves double the throughput at 400 MB/s. It's worth mentioning that C-Pack's implementation targets ASIC flow, crucial for cache compression.

- **Implications on Prior Cache Compression Work:**

Many prior studies on cache compression assumed the availability of lossless algorithms with minimal latency and hardware overheads. However, these assumptions lacked evidence or rigorous analysis. Previous research often favored cache line compression ratio over effective system-wide compression ratio, favoring algorithms producing smaller compressed line sizes. C-Pack addresses these limitations by optimizing performance, area, and power consumption while maintaining an effective system-wide compression ratio. Its success validates conclusions drawn in previous research on cache compression, provided similar characteristics to C-Pack are upheld in the compression algorithm used.

4. Conclusion

This study presents and evaluates a novel cache compression algorithm specifically designed to address the distinctive constraints of its application domain. By utilizing techniques such as pattern matching and partial dictionary coding, our algorithm facilitates parallel compression of multiple words while maintaining favorable dictionary match probabilities. Achieving an effective system-wide compression ratio of 61% and a maximum decompression latency of 6.67 ns in 65 nm process technology, our approach surpasses the performance of previously explored compression algorithms for this purpose. While initially tailored for online cache

compression, our hardware implementation exhibits versatility, requiring minimal adjustments to suit other high-performance lossless data compression applications.

5. References

1. A. R. Alameldeen and D. A. Wood, “Adaptive cache compression for high-performance processors,” in Proc. Int. Symp. Computer Architecture, Jun. 2004, pp. 212–223.
2. E. G. Hallnor and S. K. Reinhardt, “A compressed memory hierarchy using an indirect index cache,” in Proc. Workshop Memory Performance Issues, 2004, pp. 9–15.
3. R. Alameldeen and D. A.Wood, “Interactions between compression and prefetching in chip multiprocessors,” in Proc. Int. Symp. High-Performance Computer Architecture, Feb. 2007, pp. 228–239.
4. Moffat, “Implementing the PPM data compression scheme,” IEEE Trans. Commun. , vol. 38, no. 11, pp. 1917–1921, Nov. 1990.
5. M. Burrows and D. Wheeler, “A block sorting lossless data compression algorithm,” Digital Equipment Corporation, Tech. Rep. 124, 1994.
6. Alameldeen & Wood, D A, 2004, “Frequent pattern compression: A significance-based compression scheme for 12 caches”, Dept. Comp.Scie., Univ. Wisconsin-Madison, Tech
7. Alameldeen AR & Wood DA 2005, “Multifacet‟s general execution-driven multiprocessor rsimulator (gems) toolset”, In Computer Architecture News, pp.92-99.
8. Alghazo J, Akaaboune A & BotrosSf-lru, 2004, “Cache Replacement Algorithm‟, In Proceedings of the Records – International Workshop on Memory Technology, Design and Testing, pp. 19- 24
9. Ardsher Ahmed, Pat Conway, Bill Hughes & Fred Weber 2002, “shared memory MP systems”, In Proceeding of the 14th HotChips Symposium, pp.1-30.
10. Bardine A, Foglia P, Gabrielli G &Prete CA 2007, “Analysis of staticand dynamic energy consumption in nuca caches: Initial results”, In Proceedings of the Workshop on Memory Performance: Dealing with Applications, Systems and Architecture, pp. 105-112
11. Sameer Deshmukh, Rio Yokota, George Bosilca, (2023) “Cache Optimization and Performance Modeling of Batched, Small, and Rectangular Matrix Multiplication on Intel, AMD, and Fujitsu Processors” J. ACM, Vol. 1, No. 1,
12. K.Kalpana , B.Paulchamy , C.Natarajan J.B. Jebish Kumar, “Improved Elliptical Cryptography in FPGA Processor” IJIRT | Volume 10 Issue 1,pp 1101-1110, 2023