ISSN:0975-9646

Code Compression Algorithm for High Performance Micro Processor

S.Sekhar dileep kumar, K.Rakesh

Dept. of ECE, MVGR College of Engineering,
Vizianagaram, Andhra Pradesh, India

***Abstract*— Modern processors use two or more levels of cache memories to bridge the rising disparity between processor and memory speeds. Microprocessor designers have been torn between tight constraints on the amount of on-chip cache memory and the high latency of off-chip memory, such as dynamic random access memory. Accessing off-chip memory generally takes an order of magnitude more time than accessing on-chip cache, and two orders of magnitude more time than executing an instruction. Compression can improve cache performance by increasing effective cache capacity and eliminating misses. Computer systems and micro architecture researchers have proposed using hardware data compression units within the memory hierarchies of microprocessors in order to improve performance, energy efficiency, and functionality. However, most past work, and all work on cache compression, has made unsubstantiated assumptions about the performance, power consumption, and area overheads of the proposed compression algorithms and hardware .In this paper a lossless compression algorithm designed for fast on-line data compression, and cache compression in particular is proposed. The algorithm has a number of novel features tailored for this application, including combining pairs of compressed lines into one cache line and allowing parallel compression of multiple words while using a single dictionary and without degradation in compression ratio. The algorithm is proposed to a register transfer level hardware design, permitting performance, power consumption, and area estimation. The cache compression is evaluated using full-system simulation and a range of benchmarks. It can be shown that compression can improve performance for memory-intensive commercial workloads.**

***Index Terms*— Cache compression, effective system-wide compression ratio, hardware implementation, pair matching, parallel compression.**

I. INTRODUCTION

More time is essential to access off-chip memory time required to access generally takes an accessing on-chip cache. Hence to improve memory system efficiency cache hierarchies is been incorporated on chip, but it is constrained by die area and cost. Cache compression is one such technique; data in last-level on chip caches, e.g., L2 resulting in larger usable caches. However past work did not demonstrate whether the proposed compression and decompression hardware is appropriate for cache compression, considering the performance, area and power consumption requirements. This paper addresses the increasingly important issue of controlling off-chip communication in computer systems in order to maintain good performance and energy efficiency.

Cache compression presents several challenges. First, decompression and compression must be extremely fast: a significant increase in cache hit latency will overwhelm the advantages of reduced cache miss rate. This requires an efficient on-chip decompression hardware implementation. Second, the hardware should occupy little area compared to the corresponding decrease in the physical size of the cache, and should not substantially increase the total chip power consumption. Third, the algorithm should lossless compress small blocks, e.g., 64-byte cache lines, while maintaining a good compression ratio (throughout this paper we use the term ***compression ratio*** to denote the ratio of the compressed data size over the original data size). Conventional compression algorithm quality metrics, such as block compression ratio, are not appropriate for judging quality in this domain. Instead, one must consider the effective system wide compression ratio .This paper will point out a number of other relevant quality metrics for cache compression algorithms, some of which are new. Finally, cache compression should not increase power consumption substantially. The above requirements prevent the use of high-overhead compression algorithms such as the PPM family of algorithms [4] or Burrows-Wheeler transforms [5]. A faster and lower-overhead technique is required.

II. RELATED WORK AND CONTRIBUTIONS

Researchers have assumed the use of general purpose main memory compression hardware for cache compression. IBM's MXT (Memory Expansion Technology) [6]. It's a hardware memory compression/decompression technique that improves the performance of servers via increasing the usable size of off-chip main memory. Data are compressed in main memory and decompressed when moved from main memory to the off-chip shared L3 cache. Memory management hardware dynamically allocates storage in small sectors to accommodate storing variable-size compressed data block without the need for garbage collection. IBM reports compression ratios (compressed size divided by uncompressed size) ranging from 16% to 50%.

X-Match is a dictionary-based compression algorithm. It matches 32-bit words using a content addressable memory that allows partial matching with dictionary entries and outputs variable-size encoded data that depends on the type of match. To improve coding efficiency, it also uses a move-to-front coding strategy and represents smaller indexes with fewer bits. Although appropriate for compressing main memory, such hardware usually has a very large block which is inappropriate for compressing cache lines. It is shown that for X-Match and two variants of Lempel-Ziv algorithm, i.e., LZ1 and LZ2, the compression ratio for memory data deteriorates as the block size becomes smaller [7]. For example, when the block size decreases from 1 KB to 256 B, the compression ratio for LZ1 and X-Match increase by 11% and 3%. It can be inferred that the amount of increase in compression ratio could be even larger when the block size decreases from 256 B to 64 B. In addition, such hardware has performance, area, or power consumption costs that contradict its use in cache compression.

Other work proposes special-purpose cache compression hardware and evaluates only the compression ratio, disregarding other important criteria such as area and power consumption costs. Frequent pattern compression (FPC) [8] compresses cache lines at the L2 level by storing common word patterns in a compressed format. Patterns are differentiated by a 3-bit prefix. Cache lines are compressed to predetermined sizes that never exceed their original size to reduce decompression overhead. Based on logical effort analysis [9], for a 64-byte cache line, compression can be completed in three cycles and decompression in five cycles, assuming 12 fan-out-four (FO4) gate delays per cycle. To the best of my knowledge, there is no register transfer level hardware implementation or FPGA implementation of FPC power consumption, and area overheads are unknown. However, without a cache compression algorithm and hardware implementation designed and evaluated for effective system-wide compression ratio, hardware overheads, and interaction with other portions of the cache compression system, one cannot reliably determine whether the proposed architectural schemes are beneficial.

In this paper a lossless compression algorithm is been proposed and developed. The algorithm is named C-Pack, for on-chip cache compression. The main contributions of this work are as follows.

1) C-Pack targets on-chip cache compression. It permits a good compression ratio even when used on small cache lines. The performance, area, and power consumption overheads are low enough for practical use.
2) When cache compression algorithm is implemented using FPGA, performance and power requirements can be easily analyzed.
3) C-pack makes a pair of compressed lines to fit into a single uncompressed cache line.
4) The proposed hardware can be easily adapted to other high-performance lossless compression applications.

III. CACHE COMPRESSION ARCHITECTURE

Here consider private on-chip L2 cache is considered, because in contrast to a shared L2 cache, the design styles of private L2 caches remain consistent, when the number of processor cores increases. Fig. 1 gives an overview of a system architecture where compression is used. Processor has private L1 and L2 caches. L1 cache is subdivided into two parts to show separate code and data memory. L2 cache is unified in nature. Hence L2 cache is considered for this work. The main point that can be considered here is that no architectural changes are needed to be done in processor to implement the proposed techniques for a L2 cache.

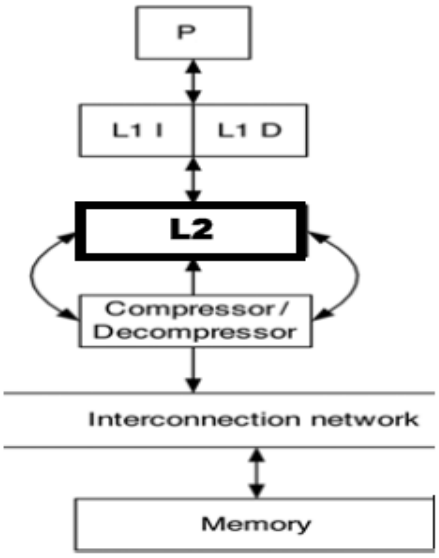

Fig-1: System Architecture in which cache compression is used

This algorithm used for compression and decompression of the data commonly found in microprocessor low-level on-chip caches, e.g., L2 caches. C-Pack which has several advantages as mentioned. Those are C-pack algorithm requires hardware that can de/compress a word in only a few CPU clock cycles. This rules out software implementations and has great influence on compression algorithm design. Cache compression algorithm is lossless to maintain correct microprocessor operation. The complexity of managing the locations of cache lines after compression influences feasibility.

It achieves a good compression ratio when used to compress data commonly found in microprocessor low-level on-chip caches, e.g., L2 caches. Its design was strongly influenced by prior work on pattern based partial dictionary match compression [11]. However, this prior work was designed for software based main memory compression and did not consider hardware implementation. C-Pack achieves compression by two means: (1) it uses statically decided, compact encodings for frequently appearing data words and (2) it encodes using a dynamically updated dictionary allowing adaptation to other frequently appearing words.

HFFFFFFF	3527894E	000756AB	12345678	AAAAAAAA	12340000	BBBB2022	VMVGRCEV

DICTIONARY

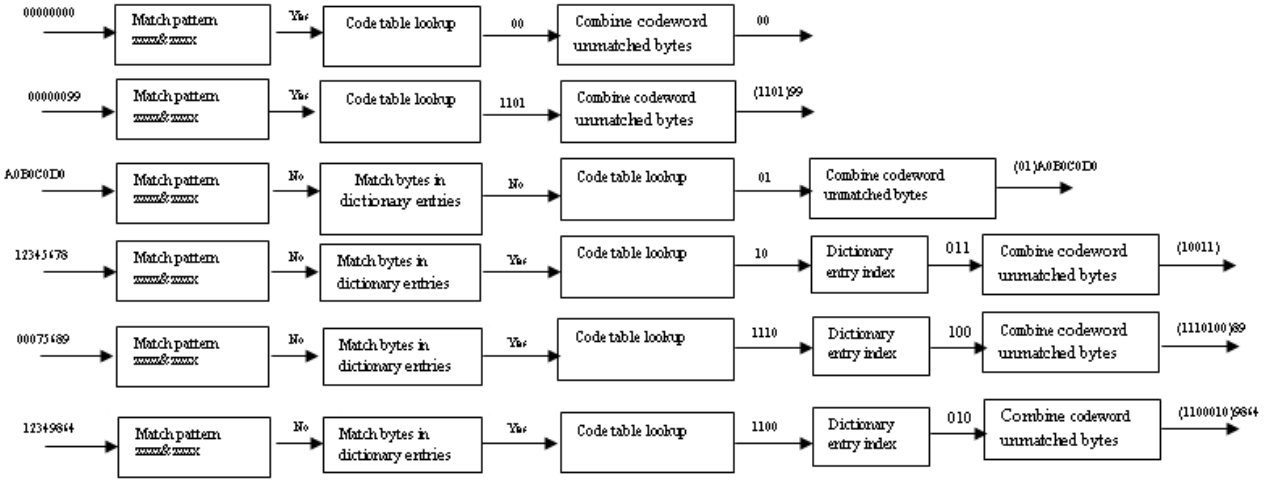

Fig-3: Example on Cache Compression

The dictionary supports partial word matching as well as full word matching.

i. C-Pack Compression Algorithm

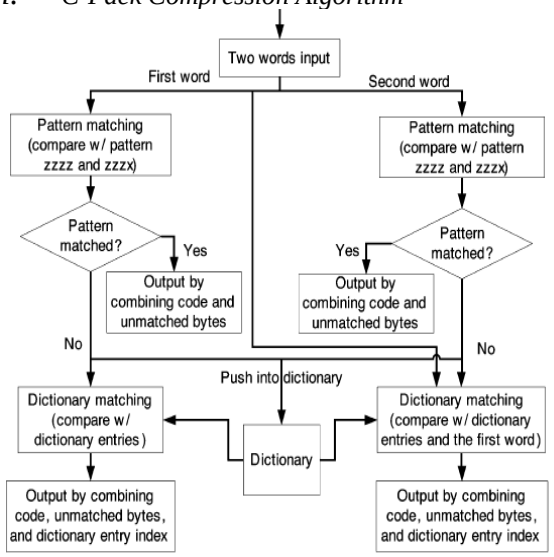

Fig 2: C-Pack Compression

The patterns and coding schemes used by C-Pack are summarized in Table I. The 'Pattern' column describes frequently appearing patterns, where 'z' represents a zero byte, 'm' represents a byte matched against a dictionary entry, and 'x' represents an unmatched byte. In the 'Output' column, 'B' represents a byte and 'b' represents a bit. The C-Pack compression algorithms are illustrated in Fig. 2. Here use an input of two words per cycle as in Fig. 2. However, the algorithm can be easily extended to cases with one, or more than two, words per cycle. During one iteration, each word is first compared with patterns "zzzz" and "zzzx". If there is a match, the compression output is produced by combining the corresponding code and unmatched bytes as indicated in Table I. Otherwise; the compressor compares the word with all dictionary entries and finds the one with the most matched bytes. The compression result is then obtained by combining code, dictionary entry index, and unmatched bytes, if any. Words that fail pattern matching are pushed into the dictionary. Fig. 3 shows the compression results for several different input words. In each output, the code and the dictionary index, if any, are enclosed in parentheses. Although we used an 8-word dictionary in Fig. 3 for illustration, the dictionary size is set to 64 B in our implementation. Note that the dictionary is updated after each word insertion, which is not shown in Fig. 3.

Code	Pattern	Output	Length(b)
00	zzzz	(00)	2
01	xxxx	(01)BBBB	34
10	mmmm	(10)bbbb	6
1100	mmxx	(1100)bbbbBB	24
1101	zzzx	(1101)B	12
1110	mmmx	(1110)bbbbB	16

Table-I: Pattern Encoding for C-pack

The Compression simulation results for the given original data are shown in the figure 4.

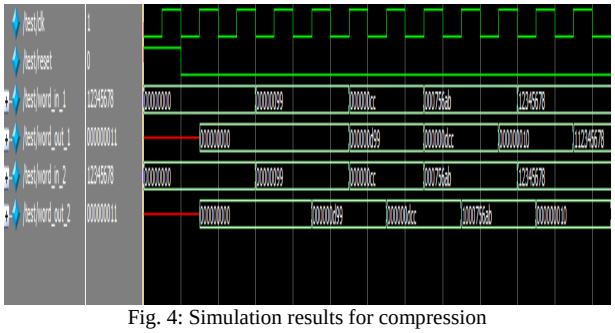

Fig. 4: Simulation results for compression

Parallel design allows an efficient hardware implementation, in which pattern matching, dictionary matching, and processing multiple words are all done simultaneously.

ii. C-Pack Decompression Algorithm

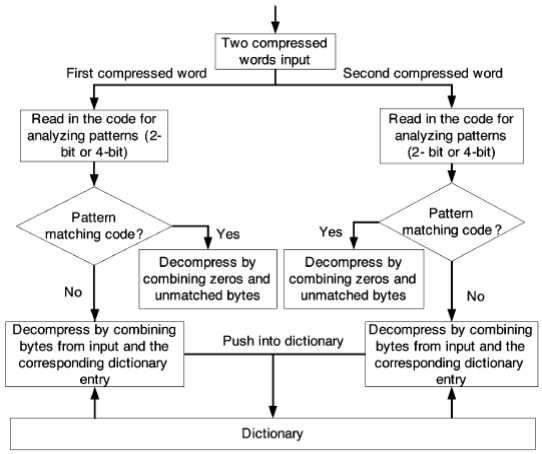

Fig 5: C-Pack Decompression

iii. Effective System-Wide Compression Ratio

During decompression, the decompression first reads compressed words is shown in figure 5.It extracts the codes for analyzing the patterns of each word, which are then compared against the codes defined in Table I. If the code indicates a pattern match, the original word is recovered by combining zeroes and unmatched bytes, if any. Otherwise, the decompression output is given by combining bytes from the input word with bytes from dictionary entries, if the code indicates a dictionary match. Decompression simulation results for the give compressed data are shown in the figure 6.

The C-Pack algorithm is designed specifically for hardware implementation. It takes advantage of simultaneous comparison of an input word with multiple potential patterns and dictionary entries. This allows rapid execution with good compression ratio in a hardware implementation, but may not be suitable for a software implementation. Software implementations commonly serialize operations. For example, matching against multiple patterns can be prohibitively expensive for software implementations when the number of patterns or dictionary entries is large. C-Pack's inherently parallel design allows an efficient hardware implementation, in which pattern matching, dictionary matching, and processing multiple words are all done simultaneously.

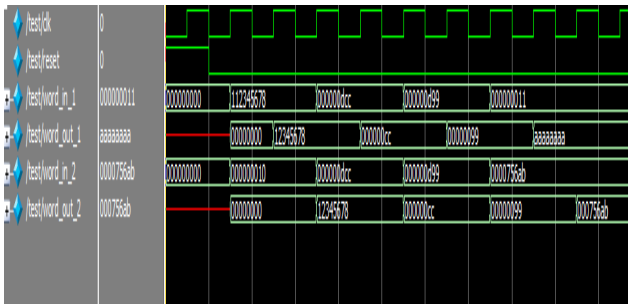

Fig. 6: Simulation results for decompression

Compressed cache organization is a difficult task because different compressed cache lines may have different lengths. Researchers have proposed numerous line segmentation techniques [2], [3], [10] to handle this problem. The main idea is to divide compressed cache lines into fixed-size segments and use indirect indexing to locate all the segments for a compressed line.

The effective system-wide compression ratio is defined as the average of the effective compression ratios of all cache lines in a compressed cache. It indicates how well a compression algorithm performs for pair matching based cache compression. The concept of effective compression ratio can also be adapted to a segmentation based approach. For example, for a cache line with 4 fixed-length segments, a compressed line has an effective compression ratio of 25% when it takes up one segment, 50% for two segments, and so on. In this paper, reduce the effective system-wide compression ratio for better high performance of microprocessor.

CONCLUSION

Code compression algorithm for high performance of microprocessor is presented to reduce the compression ratio in this work. The algorithm is based on pattern matching and partial dictionary coding. Its hardware implementation permits parallel compression of multiple words without degradation of dictionary match probability. The proposed architecture is defined in verilog HDL and simulated using Modelsim. The Code is synthesized using Xilinx XST tool and implemented using FPGA Spartan 3E starter kit. The proposed algorithm yields an effective system-wide compression ratio of 41.25%, and permits a hardware implementation with a maximum decompression latency of 6.67 ns in 65 nm process technology. It can also be used in other high-performance lossless data compression applications with few or no modifications.

REFERENCES

[1] Xi Chen, Lei Yang, Robert P. Dick,, "C-Pack: A High-Performance Microprocessor Cache Compression Algorithm" IEEE Trans. Commun. , vol. 18, no. 08, Aug. 2010.

[2] A. R. Alameldeen and D. A. Wood, "Adaptive cache compression for high performance processors," in Proc. Int. Symp. Computer Architecture, pp. 212–223, Jun. 2004

[3] E. G. Hallnor and S. K. Reinhardt, "A compressed memory hierarchy using an indirect index cache," in Proc. Workshop Memory Performance Issues, pp. 9–15, 2004,

[4] A. 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] B. Tremaine et al., "IBM memory expansion technology," IBM J. Res. Development, vol. 45, no. 2, pp. 271–285, Mar. 2001.

[7] J. L. Núñez and S. Jones, "Gbit/s lossless data compression hardware," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 11, no. 3, pp. 499–510, Jun. 2003.

[8] A. Alameldeen and D. A. Wood, "Frequent pattern compression: A significance- based compression scheme for 12 caches," Dept. Comp. Scie. , Univ. Wisconsin- Madison, Tech. Rep. 1500, Apr. 2004.

[9] I. Sutherland, R. F. Sproull, and D. Harris, Logical Effort: Designing Fast CMOS Circuits, 1st ed. San Diego, CA: Morgan Kaufmann, 1999.

[10] J.-S. Lee et al., "Design and evaluation of a selective compressed memory system," in Proc. Int. Conf. Computer Design, pp. 184–191, Oct. 1999.

[11] L. Yang, H. Lekatsas, and R. P. Dick, "High-performance operating system controlled memory compression," in *Proc. Design Automation Conf.*, Jul. 2006, pp. 701–704.