Original Article

Efficient Memory Handling For Unused Big Volume Data

[1]Thanga Anu Priya P, [1]Marys Axseelias Felcitas P, [1]Arul Selvi L

[1]Karpagamaheswari R, [2]Mr.S Cammillus,
[1]UG Student ECE, [2]Assistant Professor ECE, National Engineering College, Kovilpatti, Tamil Nadu, India.
[1] annierajo02@gmail.com, [2]cammilus@nec.edu.in
DOI: 10.47750/pnr.2023.14.04.33

Abstract

Memory usage has significantly increased in recent years. Many information are redundant and inactive such data is removed, compressed, and relocated to backups, a front end with a lot of RAM can be employed. Closed-circuit television (CCTV), sometimes known as video surveillance, is the most widely used technology in the security sector. CCTVs are present in a variety of locations, both public and private. The storage space used by the video is one of the trickiest issues with installing CCTV cameras. Hard drives and other secondary storage systems are where video is often kept. Thus, to conserve storage space, compression techniques are applied. High-quality videos are kept at the front end of the memory, which can be divided into three divisions. The backup memory is relocated at a low resolution of 50%.This will allow the user to have increased stored content for quite periods. To free up memory space, a suitable memory compression algorithm is employed. The project is implemented using Verilog code. The project is simulated using the model Sim platform.

INTRODUCTION

With a good compression technique, there will soon be excessive memory requirements because every component needs a networking stack and a memory component. Information kept in memory. Fig (1) shows the block diagram of compression and decompression procedure.

Duplicated dataUnused data Large-scale data

The amount of memory needed to store this data is tremendous. Systems must automate and control the management of Big Volume and Unused Data. Data must be of a certain size. Consequently, extra data can be kept in memory. One of the main determining elements in the design of embedded system. The size of program is therefore constrained by memory. [1].To fix the problem code compression techniques are used to make the programme smaller, Instead of displaying information in its original form, compression represents it in a condensed version. With the amount of data being stored growing, finding enough space for information retrieval and storage in the compressed area has become a top priority. The total number of bits required to represent some information will be reduced through the process of compression. It is possible to compress data using a variety of different algorithms. Because they offer both a quick decompression mechanism and a good compression ratio,dictionary-based code compression algorithms are widely used. By taking mismatches into account, recently put forth strategies enhance dictionary-based compression. To produce instruction matches, the essential concept is to remember a few bit places. The quantity of bit changes required for compression determines how effective these strategies can be. It goes without saying that more matched sequences will be produced if more bit alterations are permitted. The disadvantage of producing more recurring instruction sequences is countered by the expense of storing the data for more bit locations. There are numerous intricate compression strategies that can result in significant code size reductions. Unfortunately, a sophisticated decompression process is required by such a compression strategy, which limits system performance in general.

$$\textit{Rate of Compression} = \frac{\textit{Size of Compressed Programme}}{\textit{Program's initial file size}}$$

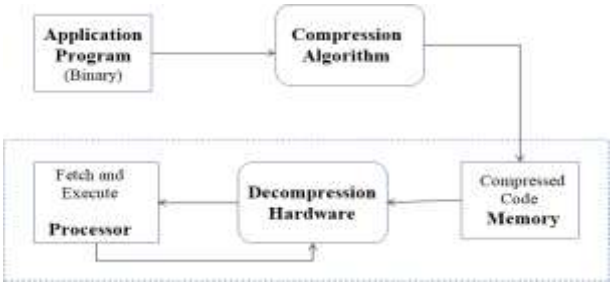


Fig 1 - Block Diagram

CODE COMPRESSION BASED ON DICTIONARIES DICTIONARY BASED APPROACH

The recurring occurrences are swapped out with a codeword that directs the reader to the dictionary's index wherethe pattern is found. Both code words and uncompressed instructions are included in this compact program. Fig(2) shows the compression pattern. The ratio [9] will be approximately 98.9% [6]. If we give 120 bits as input we will get approximately 60 bits as output so finally we get the compressed output.

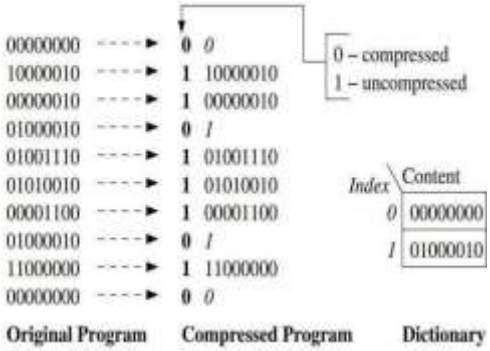


Fig 2- Compression Pattern

IMPROVED DICTIONARY BASED APPROACH

By taking mismatches into account, recently developed algorithms enhance the traditional dictionary- based compression method. This section's Fig. (3) illustratesthe dictionary-based strategy. Saving that information in the compressed programme while also updating the dictionary.

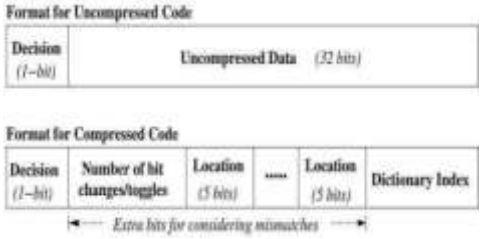


Fig 3 – Improved Dictionary based approach

How many bit changes are taken into account whenCompressing will affect the compression ratio.

COST BENEFIT ANALYSIS FOR CONSIDERINGMISMATCHES

Mismatched patterns can be compressed if we only take into account 2-bit alterations. The number of recurrent patterns can be increased by accounting for more mismatches, although this does not always enhance the compression ratio. The dictionary pattern is depicted in Fig. 4 here. [3] This is so that the compressed software can save the many bit positions that arerequired. It has been looked into how bit-masks might be used tocreate recurrent patterns.

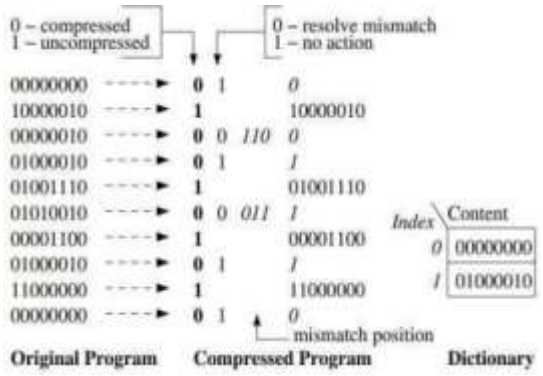


Fig 4 Dictionary Pattern

COMPRESSION ALGORITHM

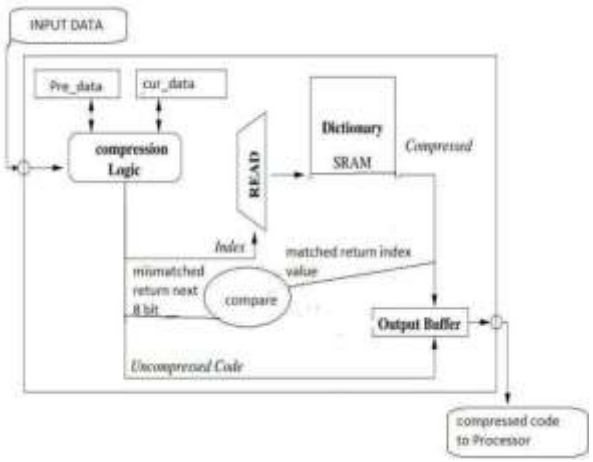


Fig 5 – Compression Algorithm

The current approach were discussed in this section, along with their shortcomings. The conventional dictionary-based approach is first described. [5] After that, we go into modern techniques that improve the traditional approach by accounting for mismatches.

The recurring occurrences are swapped out with a codeword that directs the reader to the dictionary's index where the pattern is found. Both codewords and uncompressed instructions are included in the compressed programme. Fig (5) shows the compression algorithm. The dictionary needs 16 bits, but the compressed programme needs 62 bits.

MECHANISM OF DECOMPRESSION

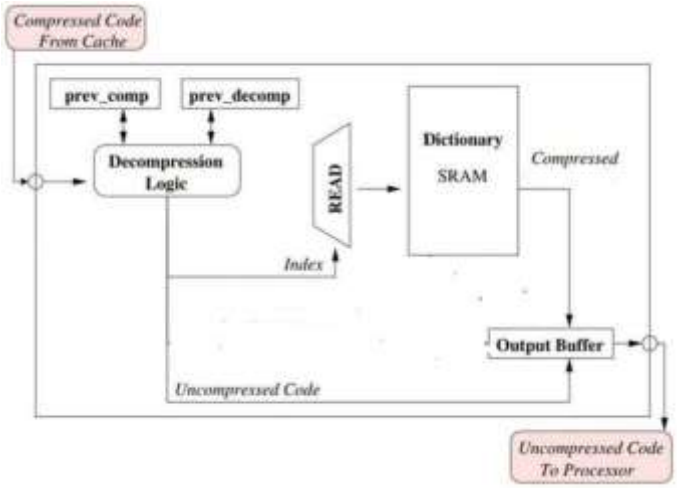


Fig 6 – Decompression Algorithm

The decompression algorithm can be used in several ways in chip designs with caches. For instance, the instruction cache and main memory can both employ the decompression hardware (pre-cache). Fig (6) here is the Decompression Algorithm.

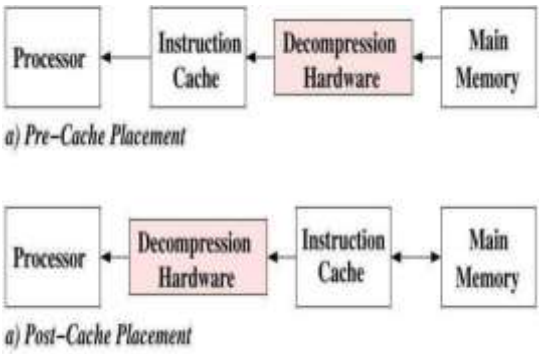
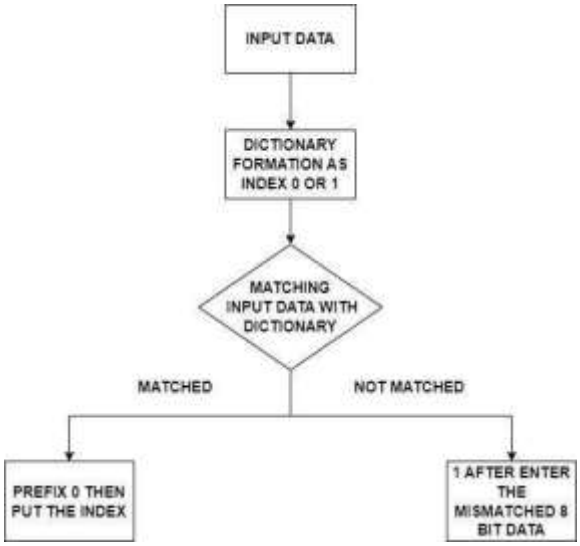


Fig 7 – Pre and Post Cache Placement

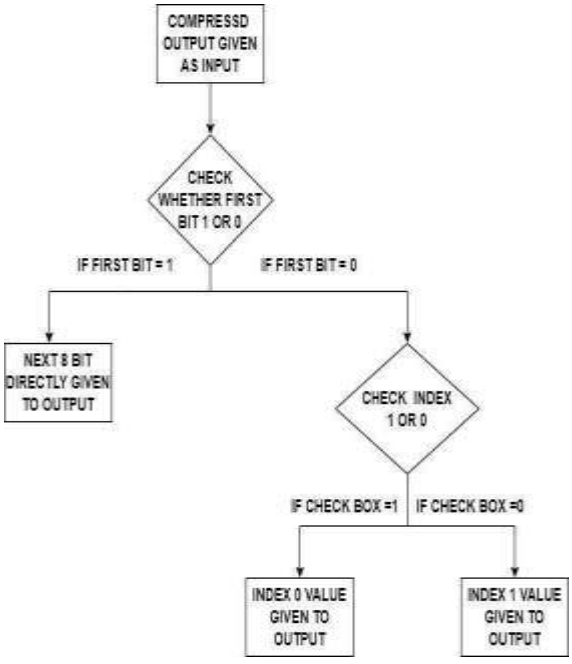
original programme will be stored in the instruction cache and the program which is compressed is being kept in main memory. Fig (7) shows the pre and post cache Placement. [4] The post- cache method depends critically on decompression (decoding) time. Our method allows for quick and easy decompression with no need to change the CPU core. [10] Ourapproach is based on a number of general criteria, such as dictionary size (index size). The design includes registers for prev comp and prev decomp. Since not all 32 bits are used by the instructions that are now being decoded, the prev comp stores the compressed data that was not used during the previous cycle. The preceding cycle's uncompressed data is stored in the prior decomp. dictionary-based code compression. Therefore, more than one instruction can be decoded simultaneously from a 32- bit stream that comprises any combination of a 12-bit code. As a result, multiple instructions can be decoded simultaneously from a 32-bit stream made up of different 12- bit code combinations. Memory is used up by both the dictionary (SRAM) and the decompression unit. However, the space needed for the dictionary is taken into account when calculating the compression ratio. As a result, while 40% code compression (or a 60% compression ratio) is given, the dictionary's space has already been taken into account.

FLOW CHART

A COMPRESSION ALGORITHM



B DICTIONARY DECOMPRESSION



RESULTS

A DICTIONARY COMPRESSION

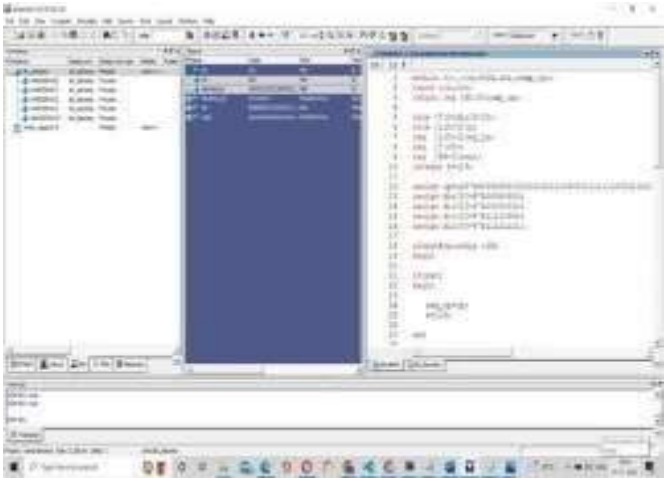


Fig 8 - Verilog Code For Compression

Fig 9 - Simulation Result For Compression

B DICTIONARY DECOMPRESSION

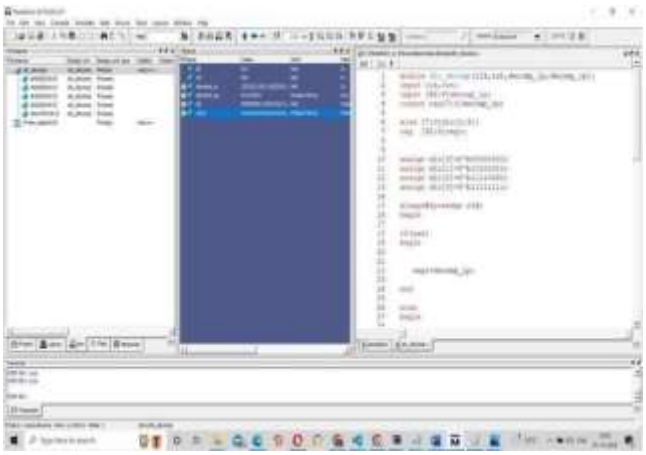


Fig 10 – Verilog code for decompression

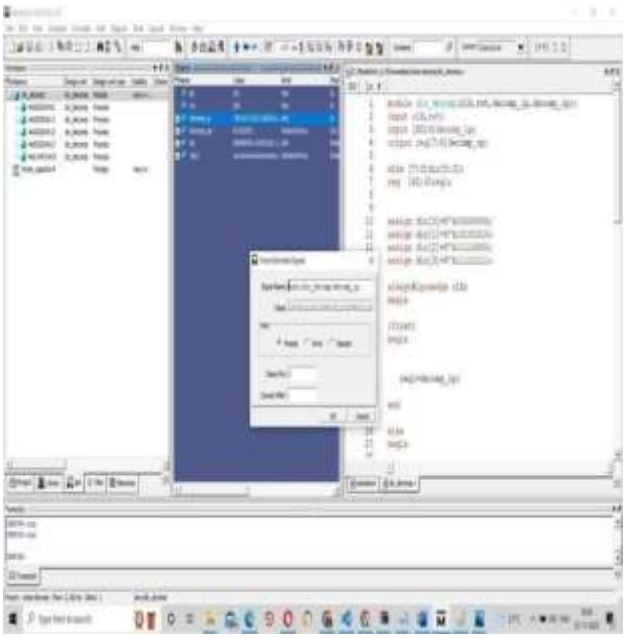


Fig 11 – Fetching Compressed input

Fig 12 – Simulation Result for decompression

CONCLUSION

Every chip has its own memory. [1 1] Now storage problem is fixed by using the compression algorithm. All these methods are very popular since they provide us a good compression. Recent methods enhance compression ratio by using bit toggling information to construct matching patterns. However, the present algorithms can match up to three slightly differently due to the lack of an effective matching system. Fig (8) to (12) given here are the results of the dictionary compression and decompression procedure. Our experimental findings indicate that our method can cut the size of the original software by up to 45%. By an average of 25%, our approach outperforms every dictionary-based technique now in use. resulting in compression [8] ratios of 58%–68%. Additionally, we suggested the layout of a quick and easy decompression unit thatcan decode one instruction each cycle in addition it is carried by parallel decompression. To demonstrate the value of our method, we used apps from diverse fields and assembled them for multiple architectures.

REFERENCES

[1] S. Seong and P. Mishra, "A bitmask-based codecompression technique for embedded systems," in Proceedings of International Conference on Computer-

Aided Design (ICCAD), 2006.

[2] S. Seong and P. Mishra, “An efficient code compression technique using application-aware bitmask and dictionary selection methods,” in Proceedings of Design Automation and Test in Europe (DATE), 2007.

[3] J. Prakash, C. Sandeep, P. Shankar and Y. Srikant, “A simple and fast scheme for code compression for VLIW processors,” in Proceedings of Data Compression Conference (DCC), 2003, p. 444.

[4] M. Ros and P. Sutton, “A hamming distance based VLIW/EPIC code compression technique,” in Proceedings of Compilers, Architectures, Synthesis for Embedded Systems (CASES), 2004, pp. 132–139.

[5] A. Wolfe and A. Chanin, “Executing compressed programs on an embedded RISC architecture,” in Proceedings of International Symposium on Microarchitecture (MICRO), 1992, pp. 81–91.

[6] H. Lekatsas and W. Wolf, “SAMC: A code compression algorithm for embedded processors,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 12, pp. 1689–1701, December 1999.

[7] H. Lekatsasand J. Henkel and V. Jakkula, “Design of an one-cycle decompression hardware for performance increase in embedded systems,” in Proceedings of Design Automation Conference (DAC), 2002, pp. 34– 39.

[8] Seok-Won Seong, Prabhat Mishra. "A bitmaskbased code compression technique for embedded systems" , 'Association for Computing Machinery (ACM)' 2007.

[9] Chetan Murthy, Prabhat Mishra. "Lossless Compression Using Efficient Encoding of Bitmasks" , 2009 IEEE Computer Society Annual Symposium on VLSI, 2009.

[10] Seok-Won Seong. "" , IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 4/2008.

[11] Dumpa Prasad, P. Rahul Reddy, B. Sreelatha, Koya Jeevan Reddy, Sudharsan Jayabalan, Asisa Kumar Panigrahy. "Recent developments in code compression techniques for embedded systems" , Materials Today: Proceedings, 2021.