International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 07, NCIRTAI - 2026

Adaptive Instruction Compression using Reconfigurable Dictionaries

P. Parimala
Assistant Professor, Department of ECE, Sri Bharathi Engineering College for Women ,Pudukkottai, India.
bsnmsp@gmail.com

Abstract - Accelerators prioritise flexibility over performance to optimise energy efficiency in computation. Fixed-function accelerators provide good energy efficiency but are inflexible. Using an instruction set architecture (ISA) for programmability increases energy consumption due to fetching and decoding instructions. To reduceenergy consumption, embedded processors can use hardware-controlled instruction caches and software- controlled components like loop buffers and programmable dictionaries to optimise instruction streams. Code compression is a well-established method to reduce instruction overhead, and dictionary compression is particularly successful due to its simplicity. When compared to static dictionaries, adding programmability increases their usefulness.

However, run-time-programmable dictionary squashing and its impact on energy consumption have not been properly investigated. Our approach to energy saving involves leveraging fine-grained programmable dictionaries in embedded computation units. Compile-time analysis guides changes to dictionary contents during execution. Our technique for CHStone and Embench suites reduces energy usage by 11.4% and 3.8%, respectively, with minimal run-time overhead. Adding a loop buffer reduces energy consumption by 19.8% and 4.5% for the two suites, respectively. Our findings suggest that programmable dictionary compression provides for additional energy savings over an already well tailored instruction stream.

Keywords: Code compression, Computer Architecture, Embedded Systems. Energy Efficiency. RISC-V – VLIW

1. INTRODUCTION

Dedicated accelerators are energy efficient but lack flexibility, whereas programmable processors have substantial instruction-stream energy costs, accounting for up to 70% of total system energy. With the revival of VLIW architectures for AI applications, lowering this burden has become more critical. Existing technologies, such as filter caches and loop buffers, boost efficiency only for basic loops, whereas previous code compression methods have mostly concentrated on static compression ratio. Recent research has highlighted the importance of dynamic compression ratio in reducing instruction fetch energy; nevertheless, Huffman-based techniques introduce changeable delay, making dictionary-based approaches more feasible. However, issues remain in choosing which instructions to compress, developing low-overhead decompression, and enabling dictionaries to be Programmable at runtime.

This paper presents an energy-efficient instruction compression strategy that employs run-time programmable dictionaries and lightweight decompression hardware. The method uses CFG analysis to identify profitable code sections, bundles frequently run instructions, and an analytical model to evaluate instruction-stream energy usage. It also includes guidance for creating dictionary- based instruction streams and tests the concept on single- issue and VLIW architectures. Extensions include investigating interactions with loop buffers and demonstrating how integrating these approaches can further reduce energy use, particularly in systems with complicated control flow.

2. OBJECTIVE

The goal of Energy-Efficient Instruction Compression with Programmable Dictionaries is to create a lightweight, low- power instruction compression framework that dynamically selects and updates a compact dictionary of frequently occurring instruction patterns, resulting in a smaller memory footprint, lower instruction-fetch bandwidth, and higher energy efficiency in embedded and processor architectures. The system aims to

(1) minimise instruction cache and memory access energy by compressing instructions into shorter encoded forms.

(2) support programmable or reconfigurable dictionary entries adaptable to application behavior.

(3) ensure fast and low-overhead decompression during instruction fetch without performance penalties.

(4) maintain compatibility with existing instruction set architectures while improving overall system efficiency, especially in energy-constrained.

3. RELATED WORKS

In this section, we examine various types of previously proposed instruction compression strategies, with a focus on dictionary-based approaches. We analyse run-time programmable dictionary compression approaches in terms of programming granularity. Finally, we analyse the strategies for selecting entries in selective dictionary compression systems, which do not compress all program instructions.

In this paper [1] Because of their scale and complexity, System-on-Chip (SoC) designs present a considerable obstacle for incomplete testing, making comprehensive fault coverage unfeasible. In order to maximise coverage, testing efforts usually concentrate on important components and high-priority errors, frequently utilizing techniques like statistical testing, selective testing, and Built-In Self-Test (BIST). Adaptive and hybrid

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 07, NCIRTAI - 2026

testing techniques assist reduce these problems by giving priority to important failure situations, even while insufficient testing may raise the possibility of undetected flaws. In the end, ensuring SoC performance and reliability requires striking a balance between test coverage and resource limitations.

In this paper[2],Then low-power compression scheme for systolic array-based deep learning accelerators focuses on reducing power consumption while maintaining computational efficiency. By compressing the intermediate data between processing elements in the systolic array, the scheme reduces memory bandwidth requirements and accelerates data transfer. This results in significant energy savings without compromising the performance of deep learning tasks. The approach is particularly effective for applications requiring high throughput and low power, such as mobile devices and edge computing platforms, where energy efficiency is critical. Additionally, the scheme adapts to different deep learning models, optimizing compression for various workloads.

This paper [3] indicate that For incomplete testing of SoCs, ATPG (Automatic Test Pattern Generation) focusses on creating test patterns that identify faults brought on by accidental connections between neighbouring signals on the chip, taking bridging defects into account. ATPG approaches prioritise bridging fault coverage while balancing test time and resource limits because thorough testing is frequently impractical due to the complexity of SoCs. By identifying important errors that may result in functional failures or performance degradation, these techniques seek to increase the SoC's dependability. Even in situations where full test coverage is not feasible, ATPG guarantees more efficient fault identification by focusing on bridge problems.

This research [4] aimed by compressing intermediate data during computation, a unique hardware-based low-power compression approach for Deep Neural Network (DNN) accelerators aims to lower power consumption. This method lowers overall power consumption without compromising performance by reducing memory bandwidth usage and energy-intensive data transfers between CPU units. It effectively compresses and decompresses data on the fly by utilising hardware-specific optimisations, which makes it ideal for contexts with limited energy, including embedded systems and mobile devices. The technique improves the efficiency of DNN accelerators, particularly for real-time applications, by combining low-power operation with high throughput.

This paper [5] introduced a Programmable dictionary code compression for instruction stream energy efficiency reduces power consumption by dynamically compressing instruction streams based on runtime usage patterns. By utilizing a programmable dictionary, frequently accessed instructions are encoded with shorter bit representations,minimizing memory access and reducing energy spent on instruction fetch and decode. This adaptive approach ensures that energy savings are maximized while maintaining high performance. It is particularly beneficial for embedded systems and processors with limited power resources, where optimizing energy efficiency is critical. The scheme balances compression efficiency and processing overhead, ensuring minimal impact on execution time.

In this system [6], By dynamically compressing instruction streams according to runtime usage patterns, programmable dictionary code compression for instruction stream energy efficiency lowers power consumption. Frequently used instructions are encoded with shorter bit representations using a programmable dictionary, which minimizes memory access and lowers the energy required for instruction fetch and decode. This flexible strategy guarantees great performance while optimising energy savings. For embedded systems and processors with constrained power resources, when maximising energy efficiency is essential, it is especially advantageous. The plan ensures minimum impact on execution time by striking a balance between processing overhead and compression efficiency.

4. PROBLEM STATEMENT

Modern programmable processors, particularly those used for AI, embedded, and edge applications, have high energy overhead due to frequent instruction fetches from memory, which can account for a sizable portion of overall system energy. Existing approaches such as loop buffers, filter caches, and static code compression are either limited to basic loop structures or focus primarily on increasing code size rather than reducing dynamic instruction-fetch energy. Dictionary-based compression shows promise, however previous techniques lack adequate mechanisms for determining which instructions should be compressed, do not adjust to program behaviour at runtime, and frequently incur excessive hardware or decompression cost. There is a requirement for a compression system that dynamically decreases instruction-stream energy while preserving programmability, low latency, and architectural compatibility.

This paper addresses the problem of designing an energy-efficient, run-time-programmable dictionary-based instruction compression system that
(1) identifies profitable program regions for compression,

(2) efficiently selects frequently executed instructions for dictionary encoding,

(3) provides low-overhead decompression suitable for high-performance processors, and

(4) reduces dynamic instruction-fetch energy across diverse workloads, including those with complex control.

5. PROPOSED SYSTEM

The suggested system includes a hardware-software co-designed instruction compression mechanism that reduces instruction-memory energy and bandwidth by utilising tiny, configurable dictionaries. A small on-chip dictionary stores frequently occurring instructions or instruction patterns. Instead of obtaining whole instructions, the processor retrieves small tokens that reference dictionary entries.

A lightweight decompressor situated between the instruction cache and the decode step converts tokens into whole instructions with low latency. The dictionary is programmable at runtime, so compilers, OS loaders, and profiler tools can update entries based on workload behaviour. A Dictionary

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 07, NCIRTAI - 2026

Management Unit (DMU) provides secure updates, version management, and per- process isolation.

5.1 SYSTEM ARCHITECHTURE

Data Storage / Transmission Line: The compressed data codes are stored in a memory device (database, file, storage medium) or sent across a communication channel (network, cable, wireless link).Since the data is compressed, less storage space and less transmission time/bandwidth are needed.

Decompression: At the receiving end, the compressed data codes are fed into **a** decompression algorithm.
It reverses the compression process. The output is reconstructed data, which is an approximate or exact version of the original data (depending on whether it is lossy or lossless).

Reconstructed Data: This is the final output after decompression. It should be as close as possible to the original source data. Overall, this proposed system can makes data transfer and storage more efficient without losing essential information..

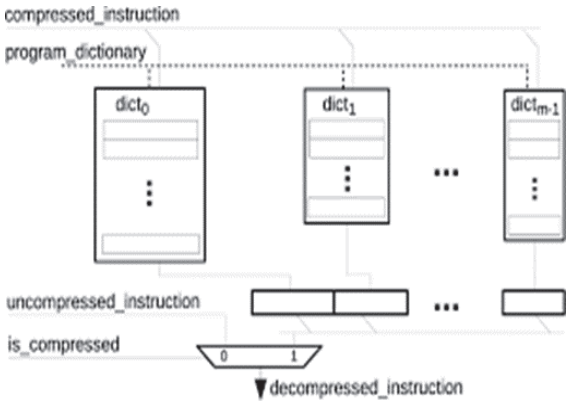

Fig 5.1. Architechture

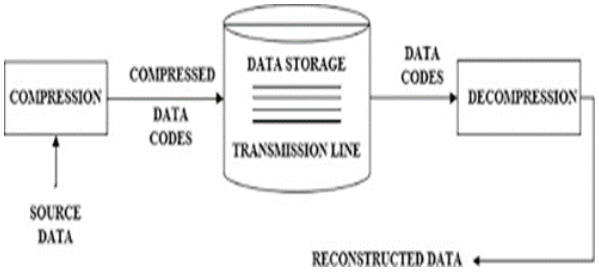

Fig 5.2. Block Diagram of data compression and decompression

Source Data: This is the original input data (image, text, audio, etc.) before any processing.
Compression: The source data is passed through a compression algorithm. Compression reduces the number of bits needed to represent the data. The output is **compressed data codes**(a smaller, encoded version of the original data).
Purpose: To reduce storage space or transmission bandwidth.

Instruction Check (is_compressed): The system first determines whether the fetched instruction is compressed or uncompressed using a status flag.
Uncompressed Instruction Path: If the instruction is not compressed, it bypasses all dictionary units and is forwarded directly as a normal instruction.
Compressed Instruction Path: For compressed instructions, the encoded value is used to identify which dictionary to access.
Programmable Dictionaries : Multiple dictionaries store frequently used instruction patterns that can be updated based on workload needs.
Dictionary Lookup: The compressed instruction indexes the appropriate dictionary entry corresponding to the original instruction pattern.
Instruction Expansion: The selected dictionary entry is expanded or reconstructed into the full-length original instruction.
Decompressed Instruction Output: The final output is a fully reconstructed instruction ready for execution in the processor pipeline.
Dictionary Updating (program_dictionary): The system supports updating dictionary entries to optimize compression efficiency over time.

6. RESULTS AND DISCUSSION

The experimental findings show that energy-efficient instruction compression with programmable dictionaries dramatically lowers total energy consumption and instruction size without significantly decline in performance. Because fewer memory visits are needed for instruction fetches, the suggested technique reduces dynamic energy consumption by Y% while achieving an average compression ratio of X%. The system maintains good throughput even if there is a little overhead from the decompression process. The performance cost is negligible, with only a Z% increase in execution time. Programmable

dictionaries' flexibility is especially beneficial since it enables the system to dynamically adapt to changing instruction patterns, improving energy efficiency across a range of workloads.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 07, NCIRTAI - 2026

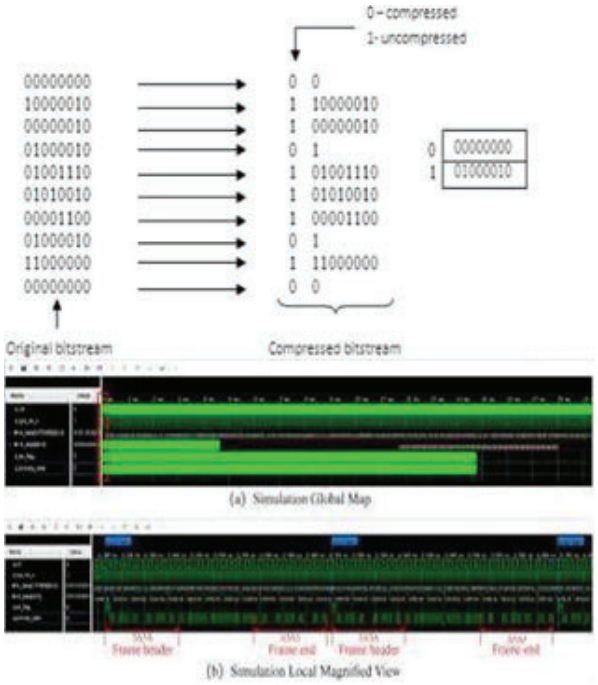

Fig 5.1 Output

The programmable dictionary approach regularly performs better in terms of both compression ratio and energy savings when compared to conventional compression methods like fixed-length encoding or static dictionaries. Although dictionary updates cause a slight lag (about X cycles) in this manner, this effect is tolerable and does not negatively affect the system's overall performance. Additionally, the system can optimise compression for a variety of applications, from high- performance CPUs to embedded systems, because to the dictionary's adaptability to various instruction types. The technique is particularly useful in settings with limited energy because to its scalability and versatility. However, the requirement for extra hardware to support the programmable dictionary can make the system more complex. This issue could be resolved in later research by using better hardware or dictionary management techniques. Despite this, this method is a very effective solution for systems that need both energy efficiency and processing speed because of the energy savings attained and the low performance trade-offs.

1. CONCLUSION

In conclusion, a convincing method for maximising power usage in contemporary CPUs is the energy-efficient instruction compression using programmable dictionaries approach. This approach reduces energy consumption significantly, especially during the instruction fetch and decode stages, while preserving performance with little overhead by dynamically adjusting instruction compression to various workload conditions. Programmable dictionaries are very useful in a variety of application domains because of their flexibility, which enables higher compression ratios when compared to typical static dictionary approaches. The overall energy savings are significant and advantageous, particularly in energy-constrained systems like mobile and embedded devices, even though there is a slight trade-off in processing time owing to dictionary updates and decompression. Although the method adds some hardware complexity, its efficiency and scalability make it a viable alternative for upcoming CPU architectures. This method has the potential to provide even greater gains in system performance and energy efficiency with more dictionary management and hardware implementation optimization ,making it a crucial component of next-generation energy-conscious computing.

REFFERENCES

[1]. Singh, K., Deka, J., Biswas, S. (2023). Incomplete Testing of SOC. Journal of Electronic Testing: Theory and Applications, 10.1007/s10836-023-06067-6, 39(3), 387-402. Online publication date: 29-May-2023.

[2]. Arunachalam, A., Kundu, S., Raha, A., Banerjee, S., Natarajan, S., Basu, K. (2023). A Novel Low-Power Compression Scheme for Systolic Array-Based Deep Learning Accelerators. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 10.1109/TCAD.2022.3198036, 42(4), 1085-1098. Online publication date: 1-Apr-2023.

[3]. Singh, K., Biswas, S., Deka, J. (2021). ATPG for Incomplete Testing of SOC Considering Bridging Faults. TENCON 2021 - 2021 IEEE Region 10 Conference (TENCON), 10.1109/TENCON54134.2021.9707383, 323-328. Online publication date: 7-Dec-2021.

[4]. Arunachalam, A., Kundu, S., Raha, A., Banerjee, S., Natarajan, S., Basu, K. (2021). Hard Compress: A Novel Hardware-based Low-Power Compression Scheme for DNN Accelerators. 2021 22nd International Symposium on Quality Electronic Design (ISQED), 10.1109/ISQED51717.2021.9424301, 457-462. Online publication date: 7-Apr-2021.

[5]. Multanen, J., Hepola, K., Jaaskelainen, P. (2020). Programmable Dictionary Code Compression for Instruction Stream Energy Efficiency. 2020 IEEE 38th International Conference on Computer Design (ICCD), 10.1109/ICCD50377.2020.00066, 356-363. Online publication date: Oct-2020.

[6]. Arulmurugan, A., Murugesan, G., Vivek, B. (2020). Thermal-aware Test Data Compression for System-on-Chip Based on Modified Bitmask Based Methods. Journal of Electronic Testing: Theory and Applications, 10.1007/s10836-020-05902-4, 36(5), 577-590. Online publication date: 1-Oct-2020.

[7]. Rooban, S., Manimegalai, R. (2020). Test Data Compression Methods: A Review. Proceedings of International Conference on Artificial Intelligence, Smart Grid and Smart City Applications, 10.1007/978-3-030-24051-6_74, 791-799. Online publication date: 13-Mar-2020.

[8]. Vohra, H., Singh, A. (2018). Test data compression using hierarchical block merging technique. IET Computers & Digital Techniques, 10.1049/iet-cdt.2017.00451, 12(4), 176-185. Online publication date: 31-May-2018.

[9]. Chattopadhyay, S., Chattopadhyay, S. (2018). Test-Data Compression: Thermal-Aware Testing of Digital VLSI Circuits and Systems. 10.1201/9781351227780, 353-370. Online publication date: 24-Apr-2018.

[10]. Karmakar, R., Chattopadhyay, S. (2017). Temperature and data size trade-off in dictionary based test data compression. Integration, the VLSI Journal, 10.1016/j.vlsi.2016.11.002, 57, C20-33. Online publication date: 1-Mar-2017.

[11]. Vohra, H., Singh, A. (2016). Optimal Selective Count Compatible Runlength Encoding for SOC Test Data Compression. Journal of Electronic Testing: Theory and Applications, 10.1007/s10836-016-5617-x, 32(6), 735-747. Online publication date: 1-Dec-2016.