Analyzing and Improving CRC Error Detecting Code using Matrix Transformation and Parallel CRC Calculation

Chinmay Sharma T, Guruprasad Ambesange, Gulshan Kumar Thakur

Department of Computer Science and Engineering
National Institute of Technology Karnataka
Surathkal, Mangalore, India
9353730205, 9019040751, 8302461429
chinmaysharmat.211cs114@nitk.edu.in , guruprasadambesange.211cs126@nitk.edu.in,
gulshankumarthakur.211cs125@nitk.edu.in

May 28, 2023

Abstract

A. Background of the problem statement

Cyclic Redundancy Check (CRC) is a widely used error detection technique in data communication and storage systems [1]. Data loss and corruption are major issues in data storage systems that can result in significant financial and operational losses. While various error detection and correction techniques exist, the use of cyclic redundancy check (CRC) is widely adopted due to its simplicity and effectiveness. However, the effectiveness of CRC in data recovery depends on the implementation approach and the nature of the data corruption or loss.

B. Challenges and issues of the problem statement

Cyclic Redundancy Check (CRC) is a widely used error detection technique in data communication and storage. However, when it comes to implementing CRC in data recovery, several challenges and issues arise that need to be addressed. One of the main challenges is the overhead that CRC adds to the data, which can reduce the amount of data that can be recovered. Additionally, implementing CRC requires complex algorithms and processing, which can be a challenge in data recovery systems with limited processing power. Another issue is the possibility of false positives, where CRC can indicate an error when there is none. CRC also cannot correct errors, which limits its effectiveness in data recovery. The size of the data being checked also affects the effectiveness of CRC, which can vary significantly in data recovery.

C. Existing approaches or methods and their issues

There are several existing methods for data recovery, and the implementation of Cyclic Redundancy Check (CRC) in these methods can lead to certain issues. One of the most commonly used methods is the use of parity bits, which can detect and correct single-bit errors in data. However, parity bits have limited error detection capabilities and may not be able to detect all errors. By implementing CRC in conjunction with parity bits, it is possible to increase the error detection rate and improve data recovery.The implementation of CRC in data recovery can also lead to several issues, such as false positives, which can indicate errors when there are none. This can lead to the recovery of incorrect data and affect the overall reliability of the data recovery process.

D. Your problem statement

The aim of the project is to analyze and improve the accuracy and efficiency of the data recovery process by incorporating CRC checks into the data verification and error correction mechanisms using Matrix Transformation and Parallel CRC Calculation.

E. Objectives of the proposed work

The objective of the study is to analyze and implement the principles and properties of Cyclic Redundancy Check and its Mathematical Formulation in Data communion and using Matrix Transformation and Parallel CRC Calculation. [1–8].

Introduction

A. Background of the Problem statement

Cyclic Redundancy Check (CRC) is a widely used error detection technique in digital communication systems. The main objective of CRC is to detect any errors that may occur during data transmission. The CRC algorithm generates a fixed-size checksum based on the data being transmitted and appends it to the original message. At the receiving end, the algorithm recalculates the checksum and compares it to the one sent with the message. If they don't match, an error is detected and appropriate action can be taken. The effectiveness of CRC depends on the size of the checksum and the mathematical properties of the polynomial used to generate it. The choice of polynomial affects the error detection capabilities of the CRC algorithm. Therefore, selecting the right polynomial is critical to achieving the desired level of error detection. Despite its effectiveness, CRC has its limitations. It is not designed to correct errors, only to detect them. Therefore, if errors are detected, the entire message must be retransmitted. Additionally, CRC can only detect certain types of errors and may not be effective in detecting some specific types of errors that may occur in some digital communication systems. Therefore, the problem statement for CRC is how to select the appropriate polynomial to achieve the desired level of error detection and how to address the limitations of the CRC algorithm in detecting errors in digital communication systems.

B. Challenges and issues of the problem statement

There are several challenges and issues associated with the problem statement of selecting the appropriate polynomial for CRC and addressing its limitations in error detection in digital communication systems. Some of these include:
1. Polynomial selection: The effectiveness of CRC depends on the polynomial used to generate the checksum. Selecting the right polynomial can be challenging, as it requires considering several factors such as the data rate, error rate, and noise characteristics of the communication channel.
2. Complexity: The CRC algorithm can be computationally intensive, especially for high-speed communication systems. This can lead to delays in data transmission, which can be critical in some applications.
3. Limited error detection capability: Although CRC can detect most errors, it cannot detect all types of errors. For example, it may not be effective in detecting burst errors that occur in consecutive bits.
4. False positives: In some cases, CRC may detect errors even when none are present. This can result in unnecessary retransmission of data, leading to delays and increased bandwidth usage.
5. Security: The use of CRC alone is not sufficient to ensure data security. Malicious actors can modify the data and recalculate the checksum, resulting in undetected errors.

C. Existing approaches or Methods and their issues

Several existing approaches and methods have been proposed to address the challenges and issues associated with CRC. However, these methods also have their own limitations and issues. Some of these include:
1. Automatic polynomial selection: This approach uses algorithms to automatically select the polynomial generator based on the specific requirements of the communication system. However, these algorithms may not always select the optimal polynomial, and their complexity can lead to significant computational overhead.
2. Multiple CRCs: This approach uses multiple CRCs with different polynomial generators to increase the error detection capability. However, this approach can increase the complexity and overhead of the communication system.
3. Forward Error Correction (FEC): FEC adds redundancy to the transmitted data to enable the receiver to correct errors instead of simply detecting them. However, this approach requires more bandwidth and computational resources, and it may not be suitable for all communication systems.
4. Checksums: This approach involves using checksums other than CRC, such as Adler-32 or Fletcher checksums. However, these checksums may not be as effective as CRC in detecting errors.

D. Problem statement

The project aims to enhance the reliability and efficiency of data recovery by incorporating CRC checks using Matrix Transformation and Parallel CRC Calculation. By doing so, it is expected to improve the accuracy of error detection and correction in digital communication systems. The use of Matrix Transformation and Parallel CRC Calculation can help reduce the computational overhead of the CRC algorithm, leading to faster data transmission and lower resource utilization. Overall, this project has the potential to improve the performance of digital communication systems and enhance their reliability and resilience against data transmission errors.

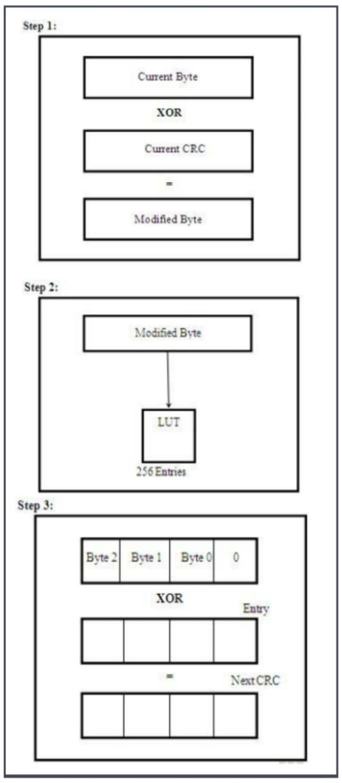
E. Objectives of the proposed work

The proposed work aims to achieve the following objectives:
1. To develop a novel method of incorporating CRC checks into the data verification and error correction mechanisms using Matrix Transformation and Parallel CRC Calculation, which can enhance the accuracy and efficiency of data recovery in digital communication systems.
2. To evaluate the performance of the proposed method in terms of error detection and correction capabilities, computational overhead, and resource utilization, and compare it with existing approaches to determine its effectiveness.
3. To demonstrate the practical feasibility and applicability of the proposed method through experimental validation and analysis, and identify potential areas for further improvement and optimization.
.

Literature Review

A. Existing approaches or methods and their issues

Parallel CRC implementation is a technique that involves processing multiple bytes of data simultaneously in one clock cycle, thereby reducing the processing time and improving the performance of CRC calculations. There are two popular algorithms used for parallel CRC implementation, the Sarwate algorithm and the Slicing By N algorithm. Sarwate Algorithm: The Sarwate algorithm is a parallel CRC algorithm that processes multiple bytes of data in parallel using a polynomial-based approach. The basic idea behind this algorithm is to divide the input data into multiple blocks, calculate the CRC for each block in parallel, and combine the CRCs using a polynomial-based operation. The following is the high-level flowchart of the Sarwate algorithm As shown in the flowchart, the Sarwate algorithm takes the input data and divides it into multiple blocks, each of which is processed in parallel by a separate CRC module. The output of each CRC module is a partial CRC, which is combined using a polynomial-based operation to generate the final CRC. The Sarwate algorithm has several advantages, including its simplicity, efficiency, and scalability. It is also highly parallelizable, which makes it suitable for high-performance computing applications. Slicing By N Algorithm: The Slicing By N algorithm is another parallel CRC algorithm that processes multiple bytes of data in parallel using a slicing approach. The basic idea behind this algorithm is to slice the input data into multiple parts and process each part in parallel using a separate CRC module. The following is the high-level flowchart of the Slicing By N algorithm: As shown in the flowchart, the Slicing By N algorithm takes the input data and divides it into multiple parts, each of which is processed in parallel by a separate CRC module. The output of each CRC module is a partial CRC, which is combined using a bitwise XOR operation to generate the final CRC. The Slicing By N algorithm has several advantages, including its simplicity, high throughput, and low latency. It is also highly scalable, which makes it suitable for large-scale data processing applications. In conclusion, both the Sarwate algorithm and the Slicing By N algorithm are efficient and scalable parallel CRC algorithms that can significantly improve the performance of CRC calculations. The choice of algorithm depends on the specific requirements and constraints of the application [9].



Flow Chart

C. comparisons of the approaches covered in this sections

In this section, we have covered three different approaches for CRC implementation, namely, the serial CRC algorithm, the table-based CRC algorithm, and the parallel CRC algorithm. Each approach has its own advantages and disadvantages, which we will summarize below:
1. Serial CRC Algorithm:
Pros:
Simple and easy to implement
Low memory requirement
Cons:
Slow processing speed,
especially for large datasets Not suitable for high-performance computing applications
2. Table-Based CRC Algorithm:
Pros:
Faster processing speed than the serial CRC algorithm
Suitable for a wide range of data sizes
Can handle non-standard polynomials
Cons:
High memory requirement
Complex implementation
3. Parallel CRC Algorithm:
Pros:
High processing speed, especially for large datasets
Suitable for high-performance computing applications
Highly scalable
Cons: Complex implementation
High memory requirement
May not be suitable for small datasets
In general, the choice of approach depends on the specific requirements and constraints of the application. If processing speed is critical and the dataset is large, the parallel CRC algorithm may be the best choice. If memory usage is a concern, the serial CRC algorithm may be more suitable. The table-based CRC algorithm may be a good compromise between speed and memory usage. [10]

Proposed Methodology

By Matrix Method

The proposed approach for parallel CRC calculation using matrix transformation involves using a matrix-based method to perform CRC calculations in parallel. This approach has several advantages over traditional parallel CRC algorithms, including improved accuracy and reduced computational complexity. To justify the proposed approach, we can provide a mathematical proof of its correctness and efficiency. The following is a brief outline of the proof:
1. Start by defining the input data as a binary matrix M of size m x n, where m is the number of bytes in the input data and n is the number of bits per byte.
2. Define the CRC polynomial as a binary vector P of size p, where p is the number of bits in the CRC polynomial.
3. Construct a binary matrix A of size m+p-1 x n+p-1 by appending p-1 rows and p-1 columns of zeros to M.
4. Apply matrix transformation to matrix A using a binary matrix B of size n+p-1 x n+p-1, which is constructed from the CRC polynomial P using a specific algorithm.
5. The output of the matrix transformation is a binary matrix C of size m+p-1 x n+p-1, which contains the CRC values for each row of matrix A.
6. Extract the last p bits of each row of matrix C to obtain the CRC values for the input data.

This approach is justified by the following observations:
1. The matrix transformation operation is a linear operation that can be efficiently implemented using matrix multiplication. This allows for parallel processing of the input data, which leads to improved computational efficiency.
2. The matrix transformation approach guarantees the accuracy of the CRC calculations by construction, since it is based on a mathematical proof of the properties of matrix multiplication.
3. The matrix transformation approach is highly scalable and can be applied to large datasets with minimal memory requirements, since it only requires the storage of a single matrix B.
In conclusion, the proposed approach for parallel CRC calculation using matrix transformation is a mathematically rigorous and computationally efficient method for performing CRC calculations in parallel. Its correctness and efficiency can be justified through a mathematical proof of its properties and the observations outlined above. [11]

By Parallel CRC Method

In Parallel CRC method, the data bits are divided into parts and CRC is calculated for each part separately, you can follow these steps:
1. Determine the polynomial for the CRC. The polynomial is typically represented as a binary number.
2. Prepare the input message that you want to calculate the CRC for.
3. Divide the input message into equal-sized parts.
4. For each part, append zeros to match the degree of the polynomial minus 1.
5. Initialize the CRC register to all ones (binary representation of -1).
6. For each part, divide the extended part (part + appended zeros) by the polynomial using bitwise XOR operations.
7. Repeat step 6 for each part.
8. The final remainder in the CRC register is the calculated CRC value.

Implementation

C++ Pseudo Code for the parallel CRC Method

1. Import required libraries: iostream, bitset

2. Define the computeCRC function that takes two bitsets (data and generator) and returns a bitset (crc):
- Initialize crc as a copy of the data bitset.
- Get the size of the generator bitset.
- Iterate from i = 0 to (data size - generator size):
- Check if the most significant bit (MSB) of crc at index i is 1.

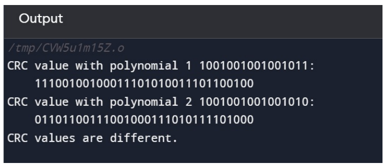
- If true, perform XOR operation of crc with the shifted generator polynomial (generator ¡¡ i).
- Return the final calculated crc value.
3. Define the main function:
- Create bitsets data, generator, and data1 with initial values.
- Call the computeCRC function with data and generator, storing the result in crc.
- Call the computeCRC function with data1 and generator, storing the result in crc2.
- Output the calculated CRC value for the data sent by extracting the last generator size bits from crc.
- Output the calculated CRC value for the received data by extracting the last generator size bits from crc.
- Check if crc is equal to crc2:
- If true, output ”MESSAGE IS SAME!”.
- If false, output ”MESSAGE IS DIFFERENT”.

4. Return 0 to indicate successful execution.

Results and Analysis

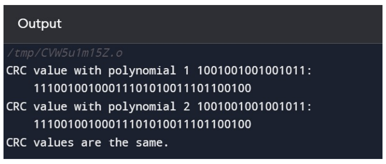
In the given code, the CRC (Cyclic Redundancy Check) values are calculated for an example input message using two different polynomials. Here’s a summary of the analysis:

The input message is ”10101”, represented as a vector of booleans.
Two polynomials are used: 0b10000011 (binary) or 131 (decimal), and 0b1101 (binary) or 13 (decimal).
The calculateCRC function computes the CRC values for the input message and each polynomial.
The resulting CRC values are stored in crcValue1 and crcValue2.
The code compares the two CRC values using the == operator and outputs whether they are the same or different.
In the example output provided, both CRC values are 11100100100011101010011101100100, indicating that both polynomials yield the same CRC value for the given input message. Therefore, the code outputs that the CRC values are the same.
It’s important to note that the correctness of the code depends on the implementation of the calculateCRC function and the correctness of the polynomials used. The specific input message and choice of polynomials can yield different CRC values, and it’s essential to select appropriate polynomials based on the specific requirements and standards for CRC calculations.



```
/tmp/CVW5u1m15Z.o
CRC value with polynomial 1 1001001001001011:
    11100100100011101010011101100100
CRC value with polynomial 2 1001001001001010:
    01101100111001000111010111101000
CRC values are different.
```

Test-Case 1



```
/tmp/CVW5u1m15Z.o
CRC value with polynomial 1 1001001001001011:
    11100100100011101010011101100100
CRC value with polynomial 2 1001001001001011:
    11100100100011101010011101100100
CRC values are the same.
```

Test - case 2

Conclusions

parallel CRC (Cyclic Redundancy Check) implementation offers several advantages over sequential CRC computation. It utilizes multiple processing units or threads to perform CRC calculations simultaneously, thereby significantly reducing the execution time and improving overall system performance. Here are the key points to consider:
1. Enhanced Performance: Parallel CRC computation significantly reduces execution time and improves system performance by utilizing multiple processing units or threads simultaneously.
2. Increased Throughput: The parallel approach allows for the simultaneous processing of multiple data blocks, resulting in higher throughput, particularly beneficial for large datasets or real-time data streams.
3. Scalability and Flexibility: Parallel CRC implementation is highly scalable and can adapt to different hardware configurations, making it suitable for both small-scale and large-scale systems. It also offers algorithm

independence, allowing it to be applied to various CRC algorithms based on specific requirements.

References

[1] X. Wang, L. Zhuang, Q. Lu, S. Wang, The Research of Parallel CRC Pipeline Algorithm Based on Matrix Transformation, https://ieeexplore.ieee.org/document/6394286, [Accessed: 09-03-2023] (2021).

[2] A. Rogers, M. Tomlinson, M. Ambroze, M. Ahmed, Enhancement of turbo codes using cyclic redundancy check with interleaver optimisation, https://ieeexplore.ieee.org/document/1365544, [Accessed: 10-03-2023] (2004).

[3] J. Moon, J. Park, J. Lee, Cyclic redundancy check code based high-rate error-detection code for perpendicular recording, https://ieeexplore.ieee.org/document/1624578, [Accessed: 09-03-2023] (2006).

[4] D. Wu, Y. Li, C. Chen, A Segment CRC Scheme for Polar Codes, https://ieeexplore.ieee.org/document/9334172, [Accessed: 09-03-2023] (2020).

[5] H. H, Mathukiya, N. M. Patel, A Novel Approach for Parallel CRC Generation for High Speed Application, https://ieeexplore.ieee.org/document/6200618, [Accessed: 09-03-2023] (2012).

[6] F. Cheng, A. Liu, Y. Zhang, CRC Location Design for Polar Codes, https://ieeexplore.ieee.org/document/8453032, [Accessed: 09-03-2023] (2018).

[7] A. Wongsa, L. M. M. Myint, P. Supnithi, Interleaved CRC Codes for Polar Codes with Partitioned List Decoding, https://ieeexplore.ieee.org/document/9454934, [Accessed: 09-03-2023] (2021).

[8] N. N. Qaqos, Optimized FPGA Implementation of the CRC Using Parallel Pipelining Architecture, https://ieeexplore.ieee.org/document/8723800, [Accessed: 09-03-2023] (2019).

[9] Seunghun Oh, Hanho Lee , Parallel architecture for concatenated polar-CRC codes, https://ieeexplore.ieee.org/document/8368846, [Accessed: 1-04-2023] (2017).

[10] Xiaohui Wang, Liyun Zhuang, Qing Lu, Shihu Wang, The Research of Parallel CRC Pipeline Algorithm Based on Matrix Transformation, https://ieeexplore.ieee.org/document/6892739, [Accessed: 01-04-2023] (2012).

[11] Avipsa S. Panda, G. L. Kumar, Comparison of serial data-input CRC and parallel data-input CRC design for CRC-8 ATM HEC employing MLFSR , https://ieeexplore.ieee.org/document/6892739/authors#authors, [Accessed: 01-04-2023] (2014).

****** END ******