Accelerating Soft-Decision Reed-Muller Decoding Using a Graphics Processing Unit

Md. Sharif Uddin1), Cheol Hong Kim2), Jong-MyonKim3)

Abstract

The Reed-Muller code is one of the efficient algorithms for multiple bit error correction, however, its high-computation requirement inherent in the decoding process prohibits its use in practical applications. To solve this problem, this paper proposes a graphics processing unit (GPU)-based parallel error control approach using Reed-Muller R(r, m) coding for real-time wireless communication systems. GPU offers a high-throughput parallel computing platform that can achieve the desired high-performance decoding by exploiting massive parallelism inherent in the algorithm. In addition, we compare the performance of the GPU-based approach with the equivalent sequential approach that runs on the traditional CPU. The experimental results indicate that the proposed GPU-based approach exceedingly outperforms the sequential approach in terms of execution time, yielding over 70x speedup.

Keywords: Real-time wireless communication, multiple bit error correction code, Reed-Muller code, GPU

1. Introduction

Wireless communications is the fastest growing field in the communications industries. Cellular systems have experienced exponential growth over the last decade. Beside the cellular systems, wireless communication systems unleashed promising arena with the diversity from wireless internet access, voice over internet protocol (VOIP), wireless video multicasting, mobile ad-hoc network (MANET), wireless sensor network (WSN) and so on. However, several technical challenges exist in designing robust and reliable wireless networks to support emerging applications. One of these challenges is to correct received error data at the receiver end because several different noise can be added to the original data in the wireless transmission medium. To correct erroneous data, an error control mechanism is necessary that can guarantee reliable communication. Therefore,
error correction codes have been used to correct the errors occurred during transmission. An error correction code includes redundant data or parity data to the original message, and then it can retrieve original message at the receiver end even if some errors are occurred during transmission, where redundant information is added to the original data packet by the encoding algorithm. The decoding algorithm then removes the redundant information after error detection and correction processes are conducted and finally retrieves the original data packet. A number of error correction codes have been proposed with the same fundamental principle.

The Reed-Muller code is one of the oldest error correcting codes used in communication systems. It is one of linear error-correcting codes. The Reed-Muller code was invented by D. E. Muller and I. S. Reed in 1954, and used by Mariner 9 to transmit black and white photographs of Mars in 1972[1]. The Reed-Muller code showed better performance than the Hamming[2] and Golay[3] codes because it allows more flexibility in the size of the code word and the number of correctable errors per code word. Whereas the Hamming and Golay codes are specific codes with particular values of q, n, k, and t, the Reed-Muller code is a class of binary codes with a wide range of design parameters. The binary Reed-Muller code is one of the most prominent codes in the coding theory. It has been extensively studied and employed for practical applications. One of the major advantages of the Reed-Muller code is its relative simplicity to encode messages and decode received data.

A majority-logic decoder for the Reed-Muller code was first proposed by Reed[4]. The Reed algorithm consists of r+1 decoding steps, where majority voting is performed. To make the algorithm more error tolerant, the value of r increases. Thus, computational complex of the decoder is very high even if it takes an advantage of the cyclic structure of the shortened Reed-Muller code[5]. Several recursive algorithms were developed that allow only O(min(r, m-r)+2m) operations[6][7]. Although the number of operations can be reduced, all these operations need to be executed sequentially. Therefore, this algorithm still requires high computational time. This has motivated research on parallel processing architectures including graphics processing units (GPUs)[8-10].

This paper proposes a GPU-based error correction mechanism using the Reed-Muller code[11-13] to support multiple bit errors in the packet level[14]. This validates the effectiveness of the proposed approach using a compute unified device architecture (CUDA)[14] enabled NVIDIA GeForce 210 graphics card. In addition, this paper compares the performance of the proposed GPU-based approach with the equivalent sequential algorithm that runs on the traditional CPU.

The rest of the paper is organized as follows. Section 2 describes the proposed GPU-based approach for accelerating the Reed-Muller code, and Section 3 describes the Reed-Muller decoder implementation on GPU. Section 4 presents experimental results and analyses. Finally, Section 5 concludes the paper.
2. Proposed Approach

Fig. 1 shows an overview of the proposed error control approach. In the proposed GPU based error control model, the sender is responsible for encoding the original using a Reed-Muller encoding scheme, and the encoded information propagates through the wireless medium to the receiver. The receiver employs a GPU to execute the Reed-Muller majority logic based decoding algorithm upon the received packets for faster error correction, resulting in retrieving the original message.

![Fig. 1] Overview of the proposed error correction approach.

For the experiment and performance evaluation, we implement the Reed-Muller decoder with R(1, 3) on a CPU and a GPU.

2.1 R(1,3) Overview

For better understanding of the Reed-Muller code, an example of the first order Reed-Muller code is introduced as follows:

Definitions: The 0th order Reed-Muller code, R(0, m), is defined as the repetition code {0, 1} of length 2m.

The first order Reed-Muller Code, R(1, m), is a binary code can be defined as the following recursive equations (1) and (2) for all integers m ≥ 1.

\[
R(1, 1) = 00.01.10.11. 
\]

(1)

For \( m > 1 \),

\[
R(1, m) = (u, u), (u, u + 1) : u \in R(1, m - 1) \text{ and } 1 = \text{ all } 1 \text{ vector} 
\]

(2)

Thus,
\[ R(1,2) = \begin{bmatrix} 0000, 0101, 1010, 1111 \\ 0011, 0110, 1001, 1100 \end{bmatrix} \]

and

\[ R(1,3) = \begin{bmatrix} 00000000, 00001111, 01010101, 01011010 \\ 10101010, 10100101, 11111111, 11110000 \\ 00110011, 00111100, 01100110, 01101001 \\ 10011001, 10010110, 11001100, 11000011 \end{bmatrix} \]

For any \( r \geq 2 \), the \( r \)th order Reed-Muller code \( RM(r,m) \) is defined recursively by the equation (3)

\[ RM(r,m) = \begin{cases} Z_2^{2^r} & \text{if } m = r \\ \{ (u, u + v) : u \in (r, m - 1) \} & \text{if } m > r \end{cases} \]  

(3)

Dimensions: The Reed-Muller \( R(r,m) \) code is a linear block code with a block length \( n=2m \), a message length \( k = 1 + \binom{m}{1} + \binom{m}{2} + \cdots + \binom{m}{r} \), minimum weight or minimum distance \( d=2m-r \), and rate \( k/2m \). Hence, \( R(r,m) \) guarantees error correction up to \( \frac{d}{2} - 1 \). Thus, for \( (1, 3) \), \( n=4, k=8 \), and it guarantees one bit error correction.

### 2.2 Generator Matrix

A generator matrix of \( R(1,1) \) is \( G_1 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \). If \( G_m \) is a generator matrix for \( R(1,m) \), then the generator matrix for \( R(1, m+1) \) is \( G_{m+1} = \begin{bmatrix} G_m & G_m \\ 00..0 & 11..1 \end{bmatrix} \). Thus, the generator matrix \( G \) for \( R(1, 3) \) is as follows:

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\]

### 2.3 Encoding

Encoding a message using the Reed-Muller code \( R(r, m) \) is straightforward. A message is sent in blocks of length \( k \). Let \( m=(m_1, m_2, m_3, \cdots, m_k) \) be a block. Then, the encoded message \( M_c \) is \( M_c = \sum_{i=1}^{k} M_i G_i \), where
G is a row of the generator matrix.

2.4 Decoding

Reed-Muller decoding is performed in the destination end of the packet, which is an exact reverse process of the encoding done in the transmitter end. Decoding the Reed-Muller’s encoded messages is much more complex than encoding. The majority logic decoding algorithm consists of the following steps[15][16]:

**Step 1:** Choose a row in the generator matrix, G. Find 2m-r characteristics for that row and then take a dot product of each of the row with the encoded message.

**Step 2:** Take the majority of the values of the dot product and assign that value to the coefficient of the row.

**Step 3:** Repeat steps 1 and 2 for all rows except the top one. From the bottom of the matrix, multiply each row with corresponding coefficient value and add the results to form My. Add My with the encoded message and then calculate majority of the values, which gives the coefficient of the first row. The original message is generated with the coefficients from top to bottom row of the matrix.

3. GPU-based Implementation of the Decoder of Reed-Muller Code

This section presents a computationally efficient implementation of the decoding algorithm of the Reed-Muller code on a GPU. The encoded packets, namely P1, P2, ...,Pn, are primarily received in the receiver buffer and the entire task can be divided into three major steps, as shown in Figure 1: (i) encoded packet transfer from CPU to GPU, (ii) device kernel execution (DKE), and (iii) decoded packet transfer from GPU to CPU. All of these steps are performed in an optimized manner from the GPU computing viewpoint.

We design a GPU optimized algorithm for a Reed-Muller code decoder. Procedure 1 is the pseudo code of CUDA (Compute Unified Device Architecture kernel) which is a programming model created by NVIDIA.

The CPU and GPU are also called Host and Device, respectively. On the outset, the encoded packet, M, is copied to the global memory of the device. Finally, the decoded packet My is copied back to host from global memory of the device. Beside this, inside host, data are copied to shared memory of the device for better performance. Shared memory is an on-chip memory. Thus, shared memory access is much faster than global memory access. We use the shared memory in the device execution steps. In procedure 1, we use the shared memory for variables temp, mults etc. which are using for temporary steps within the device. Finally, we copy the results to the global memory so that we can access them from the host.
Procedure 1 pseudo code of kernel for decoding Reed-Muller code

Input: G, generator matrix; Message, the encoded message; n, block length; k, message length
Output: Result, the decoded text

idx := index of a thread
FOR i = 1 to n - 1
mul = ANDs of permutations of rows other than i and their inverses
temp[idx] = mults[idx] AND Message[idx]
FOR p = k/2, k/4, k/8,...,1 DO
  temp[idx] = temp[idx] XOR temp[idx+p]
FOR p = n*k/2, n*k/4, n*k/8,...,k DO
  temp[idx] = temp[idx] + temp[idx+p]
IF temp[n*k*idx] < n/2 Result[i] = 0 ELSE Result[i] = 1
FOR i = 1 to n-1 DO
  IF idx < k THEN
    my[idx] = Message[idx] XOR gm[i*k+idx%k] AND Result[(idx/k)*n + i];
  FOR p = k/2, k/4, k/8,...,1 DO
    my[idx] = my[idx] + my[idx+p];
  IF my[idx*k] ≥ k/2 THEN Result[0] = 1 ELSE Result[0] = 0

For R(1, 3), the message length m=4, the encoded message length k=8 and the size of the generator matrix G is 4x8. For the GPU-based implementation, we use 32 parallel processing units for execution. Thus, a time complexity of the majority finder is O(Log2k), and the others are O(1). Beside this, we design the algorithm with the flexibility of arbitrary large packet size. In our algorithm, there are some processing on the generator matrix. Thus, in the case of R(1, 3) for a 32-bit packet, we use 256 parallel processing units, but the required operations on the generator matrix is still 32. For the packet size of multiple of 4, our procedure uses multiple of 32 parallel processing units to maintain a maximum possible parallelism. Thus, we can use a bigger packet size depending on the available processing units in the device to get a maximum speedup.

4. Experimental Results

This section presents the performance of the proposed GPU-based Reed-Muller decoding implementation and compares the GPU-based approach with the equivalent CPU-based implementation. Table 1 shows the experimental environment to simulate and evaluate the performance of the Reed-Muller decoding algorithm on the CPU and the GPU.
4.1 Execution Time

Figures 2 and 3 show the consolidated execution time by varying the packet size for sequential and parallel Reed-Muller decoding, R(1, 3), on the CPU and the GPU, respectively. The packet length ranges from 4 bits to 1024 bits on the x-axis, and the y-axis denotes the decoding time spent in milliseconds (ms) for decoding 10Mb of encoded data. Considering the message length variation, the execution time decreases with an increase of the packet size. This is mainly due to the following two reasons. First, an inverse matrix of the generator matrix needs to prepare for decoding each packet. This is a one-time task for each packet. Thus, with an increase in the number of packets, the number of computations also increase. Second, a permutation needs to select one row of the generator matrix and one row of the inverse generator matrix.

As this is also a one-time task for each packet, an increase in the number of packets also increases the number of computations. On the other hand, in the case of the GPU-based approach, execution time decreases with the increase of the packet size. This happens for the following two reasons. First, in the GPU-based implementation, the number of parallelism increases with an increase of the packet size. Second, data transfer latency from host to device and from device to host reduces with an increase of the packet size, as mentioned in Section 3.

In Figure 2, the curve for GPU is always below the curve for CPU. This means that execution time in the GPU-based approach is always less than that of the CPU-based approach. Figure 3 shows the execution time comparison of the CPU-based approach and the GPU-based approach. The GPU-based approach is up to 70 times faster than the CPU-based approach.
Accelerating Soft-Decision Reed-Muller Decoding Using a Graphics Processing Unit

[Fig. 2] Execution time of the CPU- and GPU-based approaches at different packet sizes.

[Fig. 3] Speedup comparison of the CPU and GPU-based approaches at different packet sizes.

Table 2 shows additional data presenting speedups of the GPU-based approach over the CPU-based implementation, where speedup is the ratio of time taken by the sequential execution using the CPU to that of the parallel execution using the GPU.

\[ speedup = \frac{\text{execution time of CPU-based implementation}}{\text{execution time of GPU-based implementation}} \]

The maximum speedup is achieved using the GPU over the CPU when the packet size is 192 bits because the maximum available threads saturates near this point. For a packet size larger than 192 bits, the available threads saturate and cannot complete the execution unit in a single cycle, reducing speedup. For a packet size smaller than 192 bits, achievable parallelism is less than the maximum parallelism using the available threads, and thus speedup decreases as the packet size decreases.
5. Conclusions

The quality of service (QoS) in time sensitive networks is heavily bounded by the packet processing latency. In this paper, a GPU-based Reed-Muller decoding approach was proposed for real-time error correction. Experimental results showed that the proposed GPU-based approach outperforms the equivalent CPU-based implementation in terms of computation time, yielding a maximum 70x speedup.
Accelerating Soft-Decision Reed-Muller Decoding Using a Graphics Processing Unit

References