CROSS-COMPILATION FOR CHANNEL CODING TECHNIQUES

Shivani Pradeep Tambatkar
Dept. of Electrical Engineering
San Jose State University
San Jose, California
shivanipradeep.tambatkar@sjsu.edu

Amiraj Nigam
Dept. of Electrical Engineering
San Jose State University
San Jose, California
amiraj.nigam@sjsu.edu

***Abstract*— Over the past few years, there has been tremendous increase in wireless standards in the field of communication which resulted in compatibility issues in wireless networks. The bits of information are modulated over an Analog waveform and then transmitted via a communication channel. Due to certain channel induced impairments, there would be a mismatch between the transmitted and received signal. To avoid this, some error correction control is required which is achieved by using channel coding schemes that protect the signal from the effects of the channel and help to reduce the Bit Error Rate (BER) and Frame Error Rate (FER) and improves the reliability of information transmitted.**

The channel encoding techniques chosen for this project are Polar and Turbo codes. These codes are implemented using C++ and C language respectively after which cross-compilation is performed to analyze their performance on different hardware platforms. Cross-compilation is the process of compiling code for one computer system (often known as the target) on a different system, called the host. Traditionally, in a software development process a compiler is used to convert a plain text written in a programming language into a program that can be run. But once this program is compiled it is platform dependent which restricts its usage on multiple systems. Each programming language has its compiler which would require a lot of system space. A compiler is a software that converts a program written in a high-level language into machine language while cross-compilation is a method that will create executable code for a platform other than the one on which the compiler is running. This process benefits the user to analyze their source code on different hardware platforms.

Keywords- Turbo Code, Polar Code, Cross-compilation, Bit error rate, Signal to Noise Ratio.

I. INTRODUCTION

A compiler is a software which will convert any program written in a high-level language or source code into low-level language or machine language. The hardware knows a language that is difficult for a developer to understand and hence a program is written in a high-level language to simplify the procedure for a human. To convert a high-level language into a low-level language there are several types of compilers as described in [1].

Cross-compilation is a process of generating an executable code that can be run on machine A and another unique machine B with a different processor without the need of any additional software. It is a mechanism that will compile the source code which will be used on a different architecture instead of the architecture that it was compiled on. The most common example is a compiler that is run on a windows-based PC but can generate a code that can be run on an Android smartphone.

In order to perform cross-compilation a set of toolchains is required which is specific to the architecture. A toolchain is a complete set of debuggers + linker + library + compiler + any other tools which could be required to produce the executable. These tool chains will produce an executable and architecture-specific code out of the source code.

In this project, cross-compilation is performed for channel encoding codes namely turbo and polar codes and then run on two different hardware systems using toolchains to compare their performance. Section II gives an outline of turbo and polar codes and the existing library used to compare these codes based on their Bit Error Rate and Frame Error Rate versus Signal-to-Noise ratio. Section III provides a detailed explanation of the approach adopted to achieve cross-compilation. The experimental results are presented in section IV and an overall summary is provided in section V. Section VI briefly describes the future scope for this project.

II. LITERATURE SURVEY

Forward Error Correction (FEC) or Channel coding schemes are designed to protect the signal from the effects of channel noise and interference and to ensure that the received information is as close as possible to the transmitted information. This is done by introducing redundant bits before data transmission. This in turn helps to reduce the BER and FER to improve the reliability of information transmitted. The two important categories of FEC are Convolutional Codes and Block Codes. A block code works on fixed-size packets of bits to which redundant bits are added for error correction. Turbo codes are types of block codes. Convolutional codes deal with streams of arbitrary length whose bit encoding is influenced by preceding bits. Polar Codes are a type of convolutional codes.

A. TURBO CODE

The generic form of a turbo encoder consists of two encoders separated by the interleaver. The two encoders used are normally identical and the code is such that the output contains the input bits as well. The choice of encoders used are Recursive Systematic Convolutional (RSC) encoders which are

concatenated in parallel as shown in figure 1[2]. The two encoders are separated by a matrix interleaver which arranges data in the form of a matrix and is read row-wise as shown in figure 2 [3]. The turbo decoder technique used is maximum a posterior probability (MAP) algorithm which looks for most probable value for each received bit by calculating the conditional probability of the transition from the previous bit given the probability of received bit. The decoder 1 and 2 shown in figure 3 [2] uses a MAP algorithm after receiving the information bits.

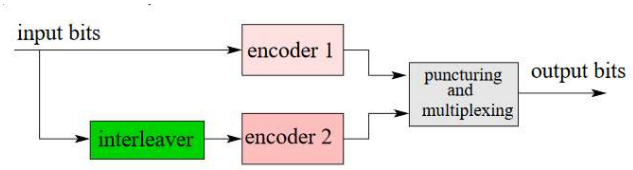

Figure 1: Encoder block Diagram for Turbo code

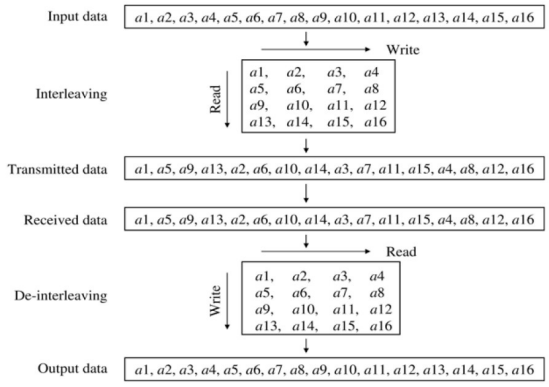

Figure 2: Interleaver block Diagram for Turbo code

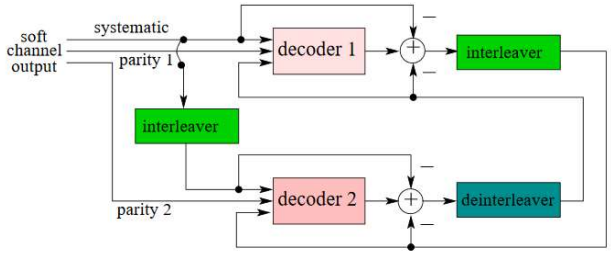

Figure 3: Decoder block Diagram for Turbo code

B. POLAR CODE

Polar Encoder [4] as shown in figure 4 consists of 3 blocks – Information block conditioning (IBC), Polar Encoder Kernel (PEK) and Encoded block conditioning (EBC)[5]. IBC interlaces the information bits with either CRC or Frozen bits or Parity check bits. The resultant is input to PEK block where the bits are split in powers of 2 and XOR operation is performed similarly to Decimation-in-Time or Decimation-in-Frequency algorithm. Finally, the EBC block will use various techniques to generate the certain encoded bits which are combined with the above output. These bits are combined and passed to the decoder block. The decoding algorithm of the receiver is in symmetry with the encoding. The polar decoder kernel operates on Successive Cancellation (SC) decoding and Successive Cancellation List (SCL) decoding.

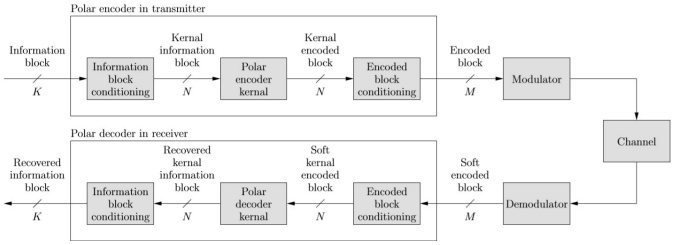

Figure 4: Block Diagram for Polar code

C. AFF3CT SIMULATOR

AFF3CT is a simulator dedicated to communication channels which focuses on the channel coding level. It is based on the Monte Carlo method wherein the transmitter will emit frames with random noise from the channel and then the receiver will decode these frames. An error occurs when the original information is different from the receiver side decoded frame.

III. PROJECT APPROACH

AFF3CT simulator is installed to generate BER and FER with respect to Eb/No ratio. The chosen channel coding techniques are Polar in C++ and Turbo codes in C. These codes are executed in a Linux environment and performance parameters are extracted. Cross-compilation is performed using QEMU/KVM tool on x86 architecture and similar performance parameters are extracted to deduce a comparison between the two architectures.

IV. EXPERIMENTAL RESULTS

A comparison of BER and FER versus Eb/No is plotted for Turbo and Polar codes using the AFF3CT simulator. Figure 5 shows this comparison for Turbo codes and Figure 6 shows a similar comparison for Polar codes.

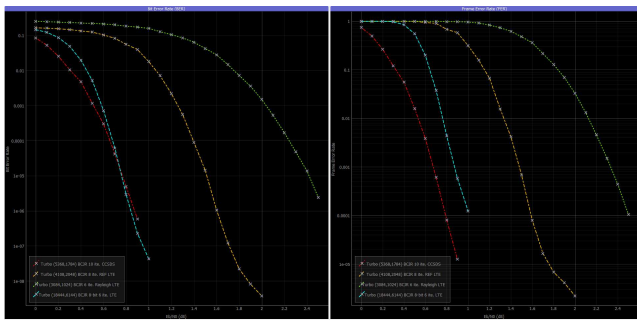

Figure 5: BER and FER versus Eb/No for Turbo Codes

Consider a case where the transmitted information bits are 1024 for Polar and Turbo codes. As seen from the graphs the block length for Turbo is 3084 whereas for Polar it is 2048 for the

same information bits. Initially, when Eb/No is low the BER and FER are high for both the codes. But gradually as the Eb/No is increased the BER and FER falls faster for Polar when compared to Turbo codes. This implies the faster execution of Polar code with respect to Turbo codes.

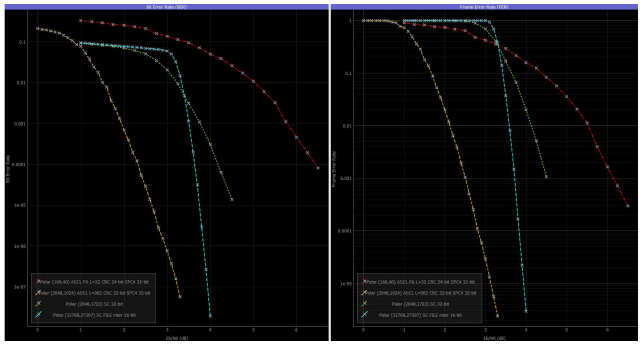

Figure 6: BER and FER versus Eb/No for Polar Codes

The environment chosen for the host computer and x86 architecture processor for cross-compilation has the following features:

Host Computer-
OS: Linux 18.04 LTS

Guest Computer (x86 Processor):
OS: Linux 15.04

x86 Processor

k	EXECUTION TIME (in secs)	CPU CYCLES (in million)	IPC
8	2.116	5	1.7
64	2.129	5	1.72
128	2.360	7	1.73
512	1.991	4	1.75
1024	1.237	4	1.8
4096	2.464	0.6	1.79

HOST COMPUTER

k	EXECUTION TIME (in secs)	CPU CYCLES (in million)	IPC
8	2.821	7	1.23
64	12.623	7	1.24
128	10.152	10	0.93
512	16.8117	9	0.77
1024	11.497	8	1.17
4096	10.875	1	0.82

Table 1. Performance metrics for Turbo Codes

Table 1 and Table 2 gives a comparison for Polar and Turbo codes (where k stands for information bits) using the following performance metrics:

1. Code Execution time: - It is the time required by CPU to execute code.

2. CPU Cycle Count: - This accounts for the basic operation such as fetch, decode and execute performed by CPU every cycle.

3. Instruction per cycle: It is the average number of instructions executed for each clock cycle.

The results show that the best architecture for Turbo codes is x86 as the execution time is less whereas Polar code is well suited for the Host Computer.

x86 Processor

K	EXECUTION TIME	CPU CYCLES (in billion)	IPC
8	4min 13.892secs	610	1.45
64	4min 15.401secs	612	1.43
128	4min 15.765secs	615	1.42
256	4min 17.604secs	617	1.42
512	4min 17.612secs	618	1.41
1024	4min 16.270secs	620	1.40
4096	4min 17.193secs	622	1.39

HOST COMPUTER

k	EXECUTION TIME	CPU CYCLES (in billion)	IPC
8	3min 40.271secs	588	1.85
64	3min 41.217secs	590	1.85
128	3min 39.903secs	588	1.85
256	3min 42.092secs	591	1.84
512	3min 42.271secs	592	1.84
1024	3min 45.993secs	599	1.82
4096	3min 43.938secs	594	1.83

Table 2. Performance metrics for Polar Codes

V. CONCLUSIONS

A comparative analysis between two sets of architecture for Turbo and Polar codes is implemented using QEMU/KVM tool in Linux Environment. The analysis showed that Turbo codes show better performance on x86 processor and Polar is well suited for the host computer. The performance metrics used for comparison were execution time, CPU cycles and Instruction per cycle. AFF3CT simulator was installed which showed that Polar codes are better performers compared to turbo codes for the same transmitted bits.

VI. FUTURE SCOPE

The current scheme does a comparison between only two architectures. This scheme can be improved by performing a similar analysis for different architectures such as ARM CORTEX-A9, ARM CORTEX-A15, etc. Additionally, comparison can be done by including other forward error correction techniques such as LDPC.

VII. ACKNOWLEDGMENT

We wish to express our sincere gratitude to our Prof. Abbas Rahimi for his constant support and encouragement which helped us to make this project successful. We would also like to thank Adrien Cassagne, Poornima Byre Gowda and Saurabh Tavildar for their repository work on this project.

REFERENCES

[1] Prajakta Pahade, Mahesh Dawale, “Introduction to compiler and its phases.” IRJET, vol. 06, issue 01, Jan 2019.

[2] southampton.ac.uk/~sqc/ELEC6214/AWCNS-L16.pdf, S.Chen, ‘Introduction to Turbo Coding and TurboDetection’ [Online].

[3] Bharath Bhambore, Poornima Byre Gowda, Prajwal Kuchchangi, Shrunga Divakar, “Turbo-Encoder-Decoder”

[4] Ajith Cyriac ; Gayathri Narayanan “ Polar Code Encoder and Decoder Implementation.", IEEE 3rd International conference on Communication and Electronic system(ICCES), 2018.

[5] G. MaunderCTO, 'The implementation challenges of polar codesRobert', AccelerComm, February 2018