Performance analysis of LDPC Decoder using OpenMP

S. V. Viraktamath
Faculty, Dept. of E&CE, SDMCET, Dharwad. Karnataka, India.

Jyothi S. Hosmath
Student, Dept. of E&CE, SDMCET, Dharwad. Karnataka, India.

Dr. Girish V Attimarad
Prof & HOD, Dept of E&CE,
Dayanand Sagar College of Engineering, Bangalore, Karnataka.India

Abstract- Digital mobile communication technologies, such as next generation mobile communication and mobile TV, are rapidly advancing. Hardware designs to provide baseband processing of new protocol standards are being actively attempted, because of concurrently emerging multiple standards and diverse needs on device functions, hardware only implementation may have reached a limit. To overcome this challenge, digital communication system designs are adopting software solutions that use central processing units (CPUs) or graphics processing units (GPUs) to implement communication protocols. In this paper, we proposed a parallel software implementation of low density parity check (LDPC) decoding algorithm, A modified version of Min-Sum algorithm (MSA) has been used for the decoding, which has the advantage of simpler computations compared to Sum-Product algorithm without any loss in performance and use a multi-core processor to achieve both flexibility and high performance. Specifically, by using Open Multi-Processing (OpenMP) to parallelize the algorithm and to implement on a multi-core processor. Processing information on H-matrices using OpenMP pragmas on a multi-core processor and compare the simulation speed against execution on single-core processor.

Keywords – LDPC code, OpenMP, MSA, Multi-core, BER

I. INTRODUCTION

In order to have a reliable communication with low power consumption over noisy channels, error correcting codes should be used. Error correcting codes insert redundancy into the transmitted data stream so that the receiver can detect and possibly correct errors that occur during transmission. Several types of codes exist. Each of which are suitable for some special applications. Researchers are searching for the best codes suitable for wireless applications. There exist a large design space with trade-offs between area of the chip, speed of decoding and power consumption. In this paper this trade-offs is addressed for a particular type of error correcting codes, namely, LDPC. These codes have proven to have very good performance over noisy channels. The main reason is that the error performance of LDPC codes can be very close to the Shannon limit. LDPC codes can be further categorized into LDPC Block Codes (LDPC-BCs) and LDPC Convolutional Codes (LDPC-CCs). In this paper, MSA is implemented in three different platforms for decoding of LDPC-BCs based on OpenMP and MATLAB programming platforms. The algorithms are executed on single core and multi-core platforms. Results are recorded and compared the simulation times. The results are compared, a significant speed and throughput is obtained using OpenMP programming.

In this paper, section I gives introduction; section II provides the Background of LDPC codes which also includes the parallelization of LDPC decoder and the introduction of OpenMP. Section III and section IV is literature survey and Methodology carried for execution of MSA LDPC decoder respectively. The results are discussed in section V.

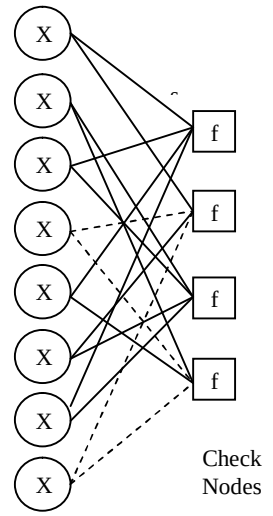
II. BACKGROUND OF LDPC CODES

Low Density Parity Check codes are a class of linear block codes corresponding to the parity check matrix H. Parity check matrix H(N-K)*N consists of only zeros and ones and is very sparse which means that the density of ones in

this matrix is very low[1]. Given K information bits, the set of LDPC code words $c \in C$ in the code space C of length N, spans the null space of the parity check matrix H in which: $cH^T = 0$, For a (Wc, Wr) regular LDPC code each column of the parity check matrix H has Wc ones and each row has Wr ones. If degrees per row or column are not constant, then the code is irregular. Some of the irregular codes have shown better performance than regular ones [2]. But irregularity results in more complex hardware and inefficiency in terms of re-usability of functional units. Code rate R is equal to K=N which means that (N / K) redundant bits have been added to the message so as to correct the errors.

A. Tanner Graphs –

LDPC codes can be represented effectively by a bi-partite graph called a "Tanner" graph. A bi-partite graph is a graph (nodes or vertices are connected by undirected edges) whose nodes may be separated into two classes, and where edges may only be connecting two nodes that residing in the same class. The two classes of nodes in a Tanner graph are "Variable Nodes or Bit Nodes (VN or BN)" and "Check Nodes (CN)". The Tanner graph of a code is drawn according to the following rule: "Check node f_j, j=1,……N-K is connected to bit node x_i, i=1, ……N whenever element h_{ij} in *H* (parity check matrix) is a *one*". Figure 1 shows a Tanner Graph made for a simple parity check matrix H. In this graph each Bit node is connected to two check nodes (Bit degree = 2) and each check node has a degree of *four*.



$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Figure 1. Tanner graph representation

B. LDPC Encoder –

The first step in designing a LDPC code is to know the preferred block length of the code, regular or irregular code, rate of the code and maximum number of decoding iterations. The Parity check matrix plays a major role in the performance the LDPC encoding/decoding. As mentioned by Gallager [1], the matrix should be very sparse. It also determines the complexity of the encoder/decoder. Depending on the platform which is going to do the encoding/decoding process, this matrix can be random or structured. Having the parity check matrix of a set of LDPC code, the corresponding Tanner graph can be drawn. To give a general perspective about encoding of LDPC codes, one might first assign each of the information bits to a Bit node in the graph, then the values of the remaining Bit nodes are determined so that all the parity check constraints satisfy. In order to put encoding process in the matrix notation, to encode a message m of K bits with LDPC codes, one might compute c = mG in which c is the N bit codeword and G_{K*N} is the generator matrix of the code in which $GH^T = 0$.

C. LDPC Decoder –

Most practical LDPC decoders use soft-decisions, because soft-decision decoders typically outperform hard-decision ones. A soft-decision decoding scheme is carried out, based on the concept of belief propagation, by passing messages, which con-tain the amount of belief for a value being between 0 and 1, between adjacent check nodes and

bit nodes. Based on the delivered messages, each node attempts to decode its own value. If the decoded value turns out to contain error, the decoding process is repeated for a predefined number of times. Typically, there are two ways to deliver messages in LDPC decoding. One is to use probabilities, and the other is to use log-likelihood ratios (LLRs). In general, using LLRs is favored since that allows us to replace expensive multiplication operations with inexpensive addition operations.

D. *Parallelization of LDPC Decoder -*

LDPC decoding algorithm is capable of correcting errors by repeatedly computing and exchanging messages. The amount of computation depends on the size of the H-matrix. However, recently published standards reveal a growing trend that the length of codewords is getting longer as the amount of data transfer is increasing. By the same token, the size of the H-matrix is increasing. The huge size causes both decoding complexity and decoding time to increase. Therefore, it is crucial to distribute the computation load evenly to multiple cores and to parallelize the computation efficiently.

LDPC decoding consists of four general operations: initialization, check node update, bit node update, and parity check. Examining the algorithm reveals that check node update can be done in parallel, since the rows are uncorrelated with each other. Also, the bit node update on each column can be processed in parallel, because an LDPC decoding algorithm has independent memory accesses among the four types of operations.

E. OpenMP-

OpenMP is a set of application program interfaces (APIs) for parallelization of C/C++ or FORTRAN programs in a shared memory multiprocessing environment. OpenMP has gained lots of attention lately as multi-core systems are being widely deployed in many embedded platforms. OpenMP is a parallelization method based on compiler directives, in which a directive will tell the compiler which part of the program should be parallelized, by generating multiple threads. Many commercial and non-commercial compilers support OpenMP directives, and thus, OpenMP can be utilized in many existing platforms. OpenMP's parallelization model is based on a fork join model. Starting with only the master thread, additional slave threads are forked on demand. All threads except for the master one are terminated when execution for a parallel region ends.

III. LITERATURE SURVEY

The LDPC code is one of the strongest error correcting codes; it is a linear block code originally devised by Gallager in 1960s [1]. However, it was impossible to implement the code in hardware that was available at that time. About 30 years later, the LDPC code was reviewed by Mackay and Neal [2]. They rediscovered the excellent properties of the code and suggested its current feasibility, thanks to the development of communication and integrated circuit technologies. Even Chung and Richardson [3] showed that the LDPC code can approach the Shannon limit to within 0.0045 dB. The LDPC code has a smaller minimum distance than the Turbo code, which was regarded as the best channel coding technique before the LDPC started to draw attention. Hence, with almost no error floor issues, it shows very good bit error rate curves. Furthermore, iterative LDPC decoding schemes based on the Sum-Product Algorithm (SPA) can fully be parallelized, leading to high-speed decoding. For these reasons, LDPC coding is widely regarded as a very attractive coding technique for high-speed 4G wireless communications. LDPC codes are used in many standards, and they support multiple data rates for each standard [9, 10]. However, it is very challenging to design decoder hardware that supports various standards and multiple data rates. Recently, Software Defined Radio (SDR) [4] baseband processing has emerged as a promising technology that can provide a cost-effective, flexible alternative by implementing a wide variety of wireless protocols in software. The physical layer baseband processing generally requires very high bandwidth and thus high processing power. Thus, multi-core processors are often employed in modern, embedded communication devices [5, 6]. Generally worldwide distributed, many of the actual parallel computing [7] platforms provide low-cost high-performance massive computation, supported by convenient programming languages, interfaces, tools and libraries. Parallel decoding of LDPC-CC is efficient in performance and reduction of time using OpenMP [8] and massively can be parallelized on GPU's [10-13].

IV. METHODOLOGY

The LDPC decoder using MSA algorithm is simulated in MATLAB with concept of executing with OpenMP and without OpenMP using MEX function. The Code rate considered is ½ rate. The results are obtained by varying SNR values and varying the number of iterations and varying in the length of the code.

A. Algorithms-

The MSA algorithm consist of four main operations, they are as follows;

- Initialization: $E_{n,m}^{(0)} = 0$ (1)
- Variable node update rule: (2)
- Check node update rule:
$$E_{n,m}^{(i)} = \prod_{n' \in N(m) \setminus n} sgn\left(T_{n',m}^{(i)}\right) \times \Phi\left(\sum_{n' \in N(m) \setminus n} \Phi\left(T_{n',m}^{(i)}\right)\right) \quad (3)$$
- Last variable node update rule:
$$T_n = I_n + \sum_{m \in M(n)} E_{n,m}^{(i-1)} \quad (4)$$

As described above, when the size of H-matrices increases, the amount of computation grows rapidly. This makes it difficult to achieve satisfactory performance in either software- or hardware-only implementations that attempt to support multiple standards and data rates. Therefore, a novel parallel software implementation of LDPC decoding algorithm is proposed that are based on OpenMP programming.

B. MSA Algorithm using OpenMP-

Parallelizing an application by using OpenMP resources often consists in identifying the most costly loops and provided that the loop iterations are independent, parallelizing them via the *#pragma omp parallel for* directive. In the algorithm it is observed that the check node update can be done in parallel since each row has no correlation with each other. Also, the bit node update on each column can be processed in parallel. As the LDPC decoding algorithm does not have dependency the parallelism can be used. The MSA algorithm implements the OpenMP for parallelizing the decoder by following steps

Step1. Include header file <omp.h>
Step2. Initialization
Step3. Compute all the messages associated to all CN's and BN's by parallelizing the code by adding OpenMP Pragma directive and which forks the N threads specified by the environment.
Step4. After computation, Master thread joins all child threads.
Step5. Perform decoding operations with N threads specified by the environment using FOR loop.
Step6. After decoding computation, BER and FER are plotted.
Step7. End.

V.RESULTS

The parameters used to analyze the performance of LDPC codes are Bit Error Rate (BER) and Frame Error Rate (FER). Parameters are defined as follows:

$$BER = \frac{Number\ of\ erroneous\ bits}{Total\ number\ of\ bits} * 100 \quad \ldots 5.1$$

$$FER = \frac{Data\ received\ with\ errors}{Total\ Data\ received} * 100 \quad \ldots . 5.2$$

In this paper, the code rate is constant and considered as ½ bit rate. A BER & FER graphs are plotted considering for different SNR in three different platforms MATLAB, C and OpenMP. Fig 2 and Fig 3 shows the BER and FER plot of three different platforms viz MATLAB, C and OpenMP for SNR varying from - 2dB to 3dB, the code length is of 1024 bits and for 15 iterations. From the figure it may be concluded that as the SNR increases the BER also increases.

When the BER results produced by the three different platforms are compared, it may be observed that the BERs are very similar regardless of the platform being used. In Fig 4 and Fig 5, the numbers of iterations are varied for constant code length and for the fixed SNR range, and then BER and FER are plotted. Table 1 shows the comparison of simulation times in three different platforms.

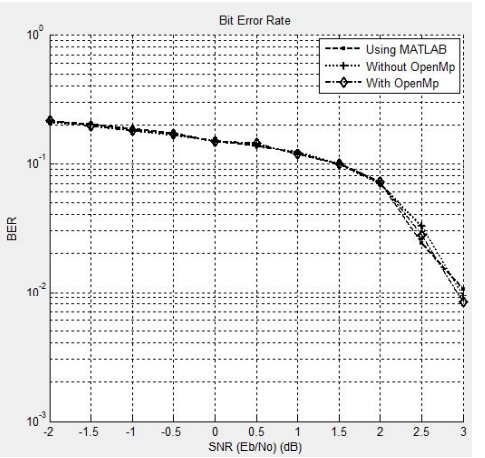


Figure 2 BER plot of three different platforms for SNR value varying from -2dB to 3dB for code length of 1024 bits.

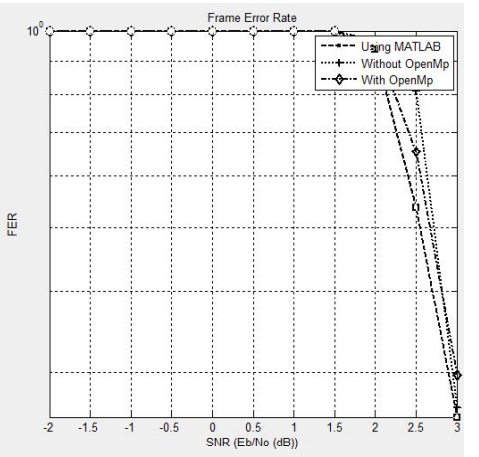


Figure 3 Frame error rate plot of three different platforms for SNR varying from -2dB to 3dB & for code length of 1024 bits

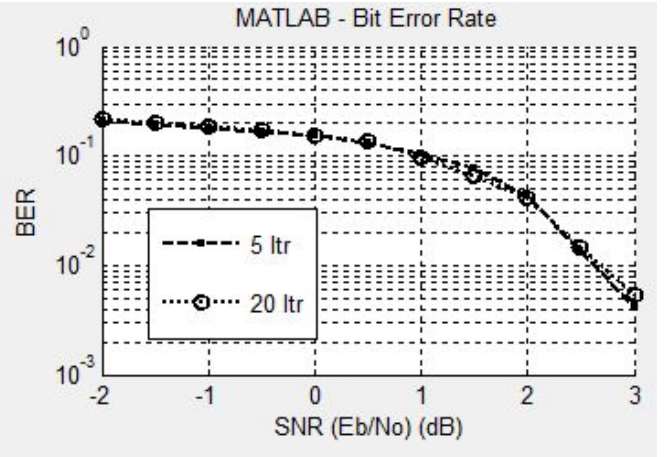


Figure 4 BER plot for iterations 5 & 20 for code length 512 bits.

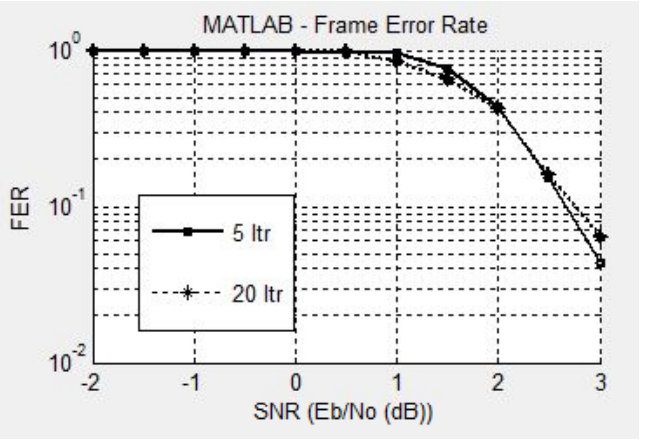


Figure 5 FER plot for iterations 5& 20 for code length 512 bits

The plots in Fig 4 and Fig 5 shows, BER and FER respectively, by varying the number of iterations with constant code length and for SNR range varying from -2 dB to 3 dB. From simulations, it is observed that varying the number of iterations yields better improvement in term of bit error rate (BER) and FER also.

Table 1: Performance Comparison between MATLAB, C and OpenM platforms for different code length

SNR Values in dB	Iterations	Code Length	MATLAB Simulation Time	C simulation time	OpenMP simulation time
3	10	128	55.916	13.7283	3.0607
		256	138.5401	15.8756	5.0555
		512	331.9808	27.0238	19.416
		1024	910.7591	60.0833	51.8255
3	15	128	125.2517	29.7734	6.7802
		256	301.1391	36.804	14.1038
		512	637.6592	47.9425	28.8675
		1024	1969.5074	111.731	90.8467
5	10	128	143.7816	35.588	8.4702
		256	345.983	44.4936	16.1668
		512	1040.0668	57.09	53.5374
		1024	3094.0234	162.234	144.532
5	15	128	322.961	70.8318	14.3021
		256	842.5557	102.1432	46.2861
		512	8256.4591	181.1307	164.6931
		1024	26603.7057	398.2875	352.260

VI.CONCLUSION

The decoding algorithm is implemented on different (MATLAB, C and OpenMP) platforms to achieve both flexibility and high performance. Specifically, using of OpenMP for parallelizing software on a multi-core processor. Test results shows that parallel software implementation of LDPC algorithms reduces the execution time thus speeding up of data processing and thereby increasing the throughput of the data. When number of iterations changes, the BER rate is nearer to Shannon limit and when matrix rate changes the error per bit is reduced and yields better results

REFERENCES

[1] R. G. Gallager, "Low Density Parity –check codes", M.I.T.Press, Cambridge, Massachusetts, 1963.
[2] D. J. MacKay and R. M. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes," *Elect. Lett.*, vol. 32, pp. 1645–1646, July, 1996.

[3] Chung, S., Forney, G., Richardson, T., and Urbanke, R. (2001), "On the Design of Low-Density Parity-Check Codes within 0.0045 dB of the Shannon Limit", *IEEE Communications Letters*, 5(2):58–60.
[4] SDR Forum, http://www.wirelessinnovation.org/
[5] L Sousa, S Momcilovic, V Silva, G Falcão, "Multi-core platforms for signal processing: source and channel coding". IEEE Press Multimedia Expo, 1805–1808 (2009)
[6] P Kollig, C Osborne, T Henriksson, "Heterogeneous multi-core platform for consumer multimedia applications". Date Conference, 1254–1259 (2009).
[7] G Falcão, L Sousa, V Silva, "Massive parallel LDPC decoding on GPU". 13th ACM SIGPLAN, 83–90 (2008).
[8] Chi H. Chan and Francis C. M. Lau, "Parallel Decoding of LDPC Convolutional Codes Using OpenMP and GPU". IEEE symposium on Computers and Communications (ISCC) (2012) 225-227.
[9] G Falcão, S Yamagiwa, L Sousa, V Silva, "Parallel LDPC decoding on GPUs using a stream-based computing approach", J Comput Sci Technol. 24(5), 913–924 (2009). DOI:10.1007/s11390-009-9266-8.
[10] G Falcão, L Sousa, V Silva, "How GPUs can outperform ASICs for fast LDPC decoding", Proceedings of the 23rd international conference on Supercomputing, 390–399 (2009)
[11] S Wang, C S, Q Wu, "A parallel decoding algorithm of LDPC codes using CUDA", Signals, Systems and Computers, 2008 42nd Asilomar Conference, 171–175 (2008)
[12] H Ji, J Cho, W Sung, "Massively parallel implementation of cyclic LDPC codes on a general purpose graphics processing unit".