Hardware Implementation of Low Density Parity Check (LDPC) Decoder

Anurag Gupta (153070050)
Guide: Prof. Madhav P. Desai
Department of Electrical Engineering
Indian Institute of Technology Bombay

***Abstract*—The objective is to implement an LDPC decoder in hardware. LDPC decoding algorithms are iterative in nature. Thus parallelization can be exploited at the hardware level to speed up the decoding process. To study the parallelization of the LDPC decoder we first constructed different low density parity check codes. Each parity check matrix related to LDPC codes can be equivalently represented by a bipartite graph, also called the Tanner graph. We partitioned the Tanner graph to study how well its edges can be parallelized. The partition efficiency of the Tanner graph corresponding to a parity check matrix represents the effectiveness of partial parallel implementation of the LDPC decoder architecture. Simulation results show that the number of parallelized edges are 78%, 88% and 86% of the number of total edges for two partitions of the Tanner graph for Gallager matrix, MacKay Neal matrix and Quasi-cyclic matrix respectively. Presently, we have implemented a one check serial decoder using bit flipping algorithm for Binary Symmetric Channel (BSC), at software level. In the future, we will parallelize it at the software level, then it will get converted to its hardware description. Then the hardware description will be mapped to FPGA and performance will be evaluated.**

I. Introduction

Low Density Parity Check (LDPC) codes are error correcting codes. These codes were first proposed by Gallager in 1962 [1]. Due to computational limitations, advantages of these codes were not exploited at that time. Then MacKay and Neal discovered a new class of block codes that outperformed state of the art turbo codes [2]. Soon the codes became popular due to their rates approaching channel capacity. The best code approaching the Shannon limit was discovered by Sae-Young Chung, for a white Gaussian noise channel threshold obtained was within 0.0045 dB of the Shannon limit with block length of 10^7 at a bit error rate of 10^{-6} [3].

As the name suggests the LDPC codes are low density parity check code. The parity check matrix of these block codes has a small number of non-zero entries. The sparseness of the matrix is the key to reduce the decoding complexity. The LDPC codes use iterative decoding that depends upon the properties of the graph representing the parity check matrix, called the Tanner graph [2]. Figure 1 shows the corresponding Tanner graph for matrix H in Figure 2.

Thus, the generation of good parity check matrix is the key for better decoding. Gallager used a regular matrix for the generation of LDPC codes [1]. With the reinvention of LDPC codes MacKay and Neal proposed random generation of matrix which performs better than Gallager matrix as it avoids cycles of 4 in the Tanner graph. In literature, it has been observed that for shorter or moderate length codes variants of Quasi-cyclic matrix is used to implement LDPC codes, as they require less space for storage and encoder structure is also less complex compared to MacKay Neal LDPC construction [4]. But, for large block length MacKay Neal codes perform better than Quasi-cyclic code.

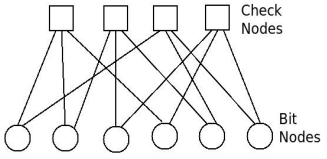

Fig. 1. Tanner graph

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

Fig. 2. Parity check matrix

Iterative decoding algorithms used for LDPC decoding are generally termed as message passing decoding. The bit flipping algorithm takes a hard decision at the received information and then principally uses majority to decode the code word. The belief propagation algorithm takes a soft decision at the received information and uses probabilistic calculation in terms of LLR (Log Likelihood Ratio) to decode the code word.

The literature survey about hardware implementation of the LDPC decoder concludes three different implementation strategies.The serial decoder is the simplest decoder. Hardware cost and complexity is very less. It consists of single check node, single bit node and memory. The bits are first passed one by one from bit side to check side and generated checks are stored in memory, then checks are passed from check side to bit side one at a time to update the bit nodes. The main drawback is this implementation is too slow [5].

The second approach is implementing a fully parallel algorithm. This is a direct implemention of the Tanner graph in hardware [5]. This increases the decoding speed, but hardware cost and implementation complexity becomes too high.

Another approach is midway between serial and parallel, called partial parallel implementation. This implementation is more flexible as trade off between speed and cost can be done. In partial parallel implementation bit nodes and check nodes are divided into several partitions. All these partitions works in parallel, but within one partition the information is transferred serially. The larger number of partitions, faster the decoder as it acts more like parallel decoder. The fewer the partition,

simpler the decoder as it acts more like serial implementation of the decoder. Thus, we can trade off between speed and cost.

Further, a reconfigurable interconnection network has to be designed to change the connection from bit to check side for different iterations in partial parallel decoder. The multistage interconnection network provides cost efficient parallel processing. Non-blocking and rearrangable networks are preferred. In blocking network connection is not always possible, whereas non-blocking network always provides a path between input to output. A rearrangable network can always provide a path, but path has to be rearranged. Lee and Ryu have used Benes network in their partial parallel LDPC decoder [6]. Benes network is a rearrangable network.

II. Partitioning

To use the partial parallel approach we propose to iteratively decompose the Tanner graph into sets of bits and sets of checks. In one iteration one set of bits is transferred towards one set of checks. It will be preferred that maximum number of the bits required for a particular set of checks are in same bit set so that check calculation can be completed for as many checks as possible. If a check set requires bits from many bit sets, then the number of iterations required to complete checks of that set will be equal to the number of bit sets required to calculate all its checks. Thus, if many check sets require bits from a large number of bit sets, then number of iterations increases and decoding time will become worse. Thus, an efficient partitioning of bit and check side of the Tanner graph is required.

The objective is to partition the bipartite graph corresponding to parity check matrix into two subsets in both bit side and check side such that most of the edges of a set of bit side relates to a particular set of check side. We call these edges as 'Through edges'. The remaining edges of the same set on the bit side that goes to different set in check side are called 'Transverse edges'. The average ratio of total number of through edges to total number of edges is the performance index for parallelization.

Suppose we have bipartite graph G, bits are B={b_1,b_2,b_3,...,b_n} and checks are C={c_1,c_2,c_3,...,c_m} and the edge between bit node b_i and check node c_j is represented as $e_{i,j}$. Now we have to partition G into four subsets B_1,B_2,C_1 and C_2 such that

- Size of set B_1 and set B_2 should be nearly equal as well as size of C_1 and C_2 should be nearly equal.
- There should be a very small number of edges between B_1-C_2 and B_2-C_1 so that the number of cross edges are minimised.

To approach this we initially find a weighted check matrix from the parity check matrix. The weighted check matrix is an incidence matrix for the check node graph. A check node graph is a weighted graph. The weight between $checknode_i$ and $checknode_j$ is the number of common bit nodes performing check on both $checknode_i$ and $checknode_j$. After forming the weighted check graph we partitioned it into two equal parts using METIS [7]. After partitioning the check set into two

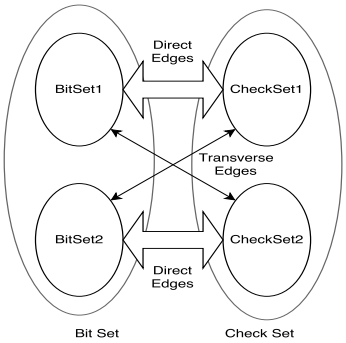

Fig. 3. Partitioning a bipartite graph

equal sets with minimum weight cut, we need to partition the bit set. To partition the bit set we take into account the number of checks associated with a bit. If number of checks associated with a bit are more in first check set then it goes to first check set else it goes to second check set. As all matrices are sparse, the number of bits divided in two sets comes out nearly equal. To make them exactly equal we force some bits to go to other set after partitioning. Then the average ratio of parallelized edges to total edges and transverse edges to total edges is calculated as a performance index of parallelization of the decoder. Figure 4 shows the block diagram for the process of partitioning the bipartite graph corresponding to a low density parity check matrix.

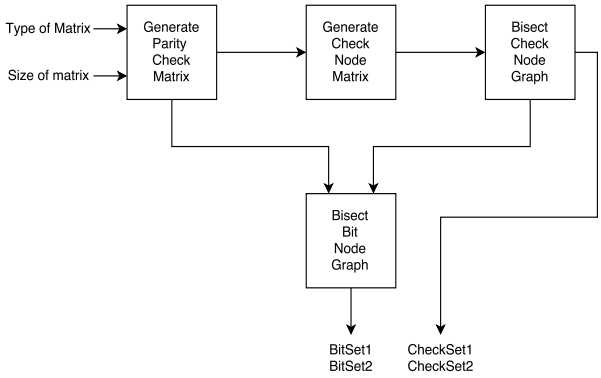

Fig. 4. Block Diagram of process flow

A. Generation of Parity Check Matrix

- Gallager Parity Check Matrix [1]: Gallager proposed parity check matrix is regular in nature. A regular matrix has a constant number of non-zero entries in a row and a column. These are represented as (n,w_c,w_r) codes where w_c= no of 1's in a column, w_r= number of 1's in a row, n=block length.
  Method of construction:
  - Divide rows in w_c sets with m/w_c rows in each set.
  - All rows of first set of rows contain w_r consecutive ones ordered from left to right.

– Every other set of row is random column permutation of first set of rows.

Example: (n,w_c,w_r)=(12,3,4); m=9

$$H = \left[\begin{array}{cccccccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ - & - & - & - & - & - & - & - & - & - & - & - \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ - & - & - & - & - & - & - & - & - & - & - & - \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

- Quasi-Cyclic (QC) parity Check Matrix [8]: QC matrix can be constructed by Quadratic Congruences (QC) Structure Method.

  Method of Construction of (j,k)-regular QC-LDPC code:
  - Construct two sequences $\{s_1, s_2, ..., s_{j-1}\}$ and $\{t_1, t_2, ..., t_{k-1}\}$, whose elements are randomly selected from GF(p), where p is prime and p>2 , $s_i \neq s_x$ & $t_i \neq t_x$ if i ≠x.
  - Now, form a preliminary matrix Y with the elements of GF(p) as follows:

$$E = \left[\begin{array}{cccc} e_{0,0} & e_{0,1} & \cdots & e_{0,k-1} \\ e_{1,0} & e_{0,1} & \cdots & e_{0,k-1} \\ \vdots & \vdots & \ddots & \vdots \\ e_{j-1,0} & e_{j-1,1} & \cdots & e_{j-1,k-1} \end{array}\right] \quad (1)$$

    where (i,j)th element of E is calculated by following quadratic congruential equation for a fixed parameter $\kappa \epsilon \{1, 2, ..., p-1\}$ and $\nu_i, \nu_j \epsilon \{1, 2, ..., p-1\}$:

$$e_{i,j} = [\kappa(s_i + t_j)^2 + \nu_i + \nu_j] \quad (2)$$

  - So the parity check matrix H is represented by $j \times k$ array of circulant permutation of identity matrix.

$$H = \left[\begin{array}{cccc} I(e_{0,0}) & I(e_{0,1}) & \cdots & I(e_{0,k-1}) \\ I(e_{1,0}) & I(e_{0,1}) & \cdots & I(e_{0,k-1}) \\ \vdots & \vdots & \ddots & \vdots \\ I(e_{j-1,0}) & I(e_{j-1,1}) & \cdots & I(e_{j-1,k-1}) \end{array}\right]$$

    Where I(x) is $p \times p$ identity matrix with row cyclically shifted right by x position.

- MacKay Neal Parity Check Matrix [9]: MacKay Neal proposed regular (n,w_c,w_r) construction of codes using a random distribution of non-zero entries. These codes have better performance for large block length compared to other codes. Method of construction:
  - Start from the first column.Place w_c 1's in the column randomly and track number of 1's in a row.
  - Repeat the process for other columns. Break only if at any point number of 1's in the row becomes greater than w_r.
  - If break occurs then go back to some columns and repeat algorithm till all columns get filled.

  MacKay Neal construction can be adapted to avoid cycles of length 4, called 4-cycles.

  Method to avoid 4-cycles:
  - Generate a preliminary parity check matrix.
  - Add 1's to check matrix in the rows that have no 1's in them, hence are redundant, or that have only one 1, in which case corresponding codeword bits will always be zero. The places in these rows to add 1's are selected randomly.
  - Choose odd number of 1's to put in a column. If preliminary parity check matrix constructed has an even number of 1's in each column, problem may occur that will cause the rows to add to zero, and hence at least one check will become redundant.
  - To eliminate situation where a pair of columns both have 1's in a particular pair of rows, which correspond to cycles of length four in the factor graph, one of the 1's involved is moved randomly within its column.

B. Generation of Check Node Incident Matrix

The weight between two check nodes is the number of common bit nodes performing Exor operation on both check nodes. Taking this into account, we followed following algorithm to convert a parity check matrix to a check node incidence matrix.

Algorithm 1 Pseudo-code for generation of check node incidence matrix

```
function W=WeightedCheckMatrix(H)        ▷ H is parity check matrix
    [m,n]=size(H)
    W=zeros(m,m);
    for i = 1 : n do
        A = find(H(:, i))
        SizeA = Size(A, 1)
        if SizeA ≠ 1 then
            for j = 1 : SizeA − 1 do
                for k = j + 1 : SizeA do
                    W(A(j, 1), A(k, 1)) =
                    W(A(j, 1), A(k, 1)) + 1
                end for
            end for
        end if
    end for
    W = W + W'
end function
```

C. Bisection of Check Node Graph

Partitioning of check node graph is done using METIS [7]. METIS is a set of serial programs for partitioning graphs, partitioning finite element meshes, and producing fill reducing

orderings for sparse matrices. The algorithms implemented in METIS are based on the multilevel recursive-bisection, multilevel k-way, and multi-constraint partitioning schemes [7]. First, the incidence matrix is converted into a corresponding graph format that can be given as input to METIS. At output we get two partitions of equal size with the minimum weight cut.

D. Bisection of Bit Node Graph

After bisection of the check set we get two sets C_1 and C_2. Now, we have to divide a partition of bit set into B_1 and B_2. A bit node corresponds to set B_1 if the number of edges between that bit node and set C_1 are more than the number of edges between bit node and set C_2. As the check set is bisected by keeping in mind that the more the number of checks relating to a bit are in same set and the matrix is sparse thus we get nearly equal bits in set B_1 and set B_2. As this bisection has to be further iterated for partitioning, we force the bisection to be equal. Forcing the bisection to be equal increases the transverse edges.

III. Results of partitioning applied to matrices

Simulation for partitioning are performed in MATLAB environment. Figure 5 shows the performance index for Gallager

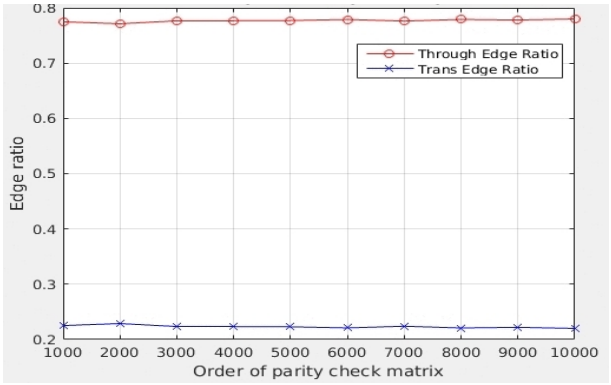

Fig. 5. Performance of Gallager matrix

matrix when we vary the order of the parity check matrix from 1,000 to 10,000. The performance index comes 0.78 for Gallager matrix. Similarly, by varying the order of the parity check matrix from 1,000 to 10,000 we get performance index of MacKay Neal matrix as 0.88, depicted in Figure 6. Quasi-Cyclic matrix can be formed only for a special set of numbers. In simulation,the order of matrix taken is 155,305,905,11555 and performance index is above 0.88 in all cases, as depicted in Figure 7.

IV. Serial LDPC decoder

We have implemented one check serial decoder in C. This is the simplest decoder that can be implemented. We have used bit-flipping decoding algorithm for a Binary Symmetric Channel (BSC) model.

In the bit-flipping decoding a hard decision is made by the detector for the received bits before passing them to decoder.

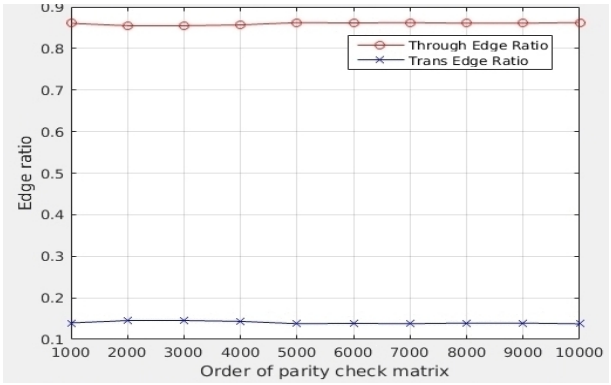

Fig. 6. Performance of MacKay Neal matrix

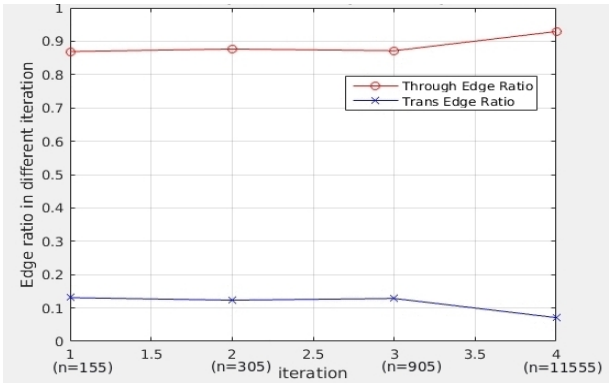

Fig. 7. Performance of quasi-cyclic matrix

Then bits are placed on the bit nodes and passed through the edges of the Tanner graph. The check node determines that its parity-check equation is satisfied if the modulo-2 sum of the incoming bit values is zero else they pass the flipped value to the corresponding bit. If the majority of the reception by a bit node are different from its present value then bit node changes (flips) its current value. This process is repeated until all of the parity-check equations are satisfied, or the decoder gives up. As LDPC matrix is sparse it is unlikely to have the same set of checks for a single bit. Still if several checks applied to single bit are incorrect then it is likely to be flipped.

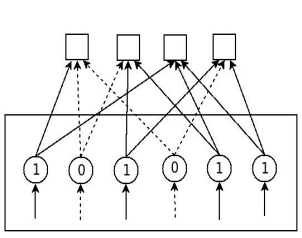

Fig. 8. Initialization

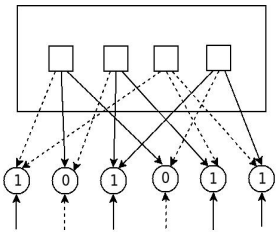

Fig. 9. Performing Checks

Figure 8,9,10,11 shows the process of bit-flipping decoding in four steps. The Tanner graph corresponds to parity check matrix depicted in Figure 2. First we take a hard decision on

Algorithm 2 Pseudo-code for one check serial decoder

```
1: function DECODEWORD=D(CompressedMatrix,CodeWord)
2:     [Colind, RowPtr] ← CompressedMatrix
3:     Bitvec ← CodeWord
4:     int itr = 0
5:     int *Bitxor
6:     bool Exorall
7:     while (itr < 100) && (CodeWord * H' == 0) do
8:         for i = 0 : m − 1 do
9:             Bitxor = Colind + Rowptr[i] − 1
10:            Exorall = 0
11:            ▷ finding exor of all bits connected to a check
12:            for j = 1 : RowPtr[i + 1] − RowPtr[i] do
13:                Exorall = Exorall ⊕ Bitvec[*(Bitxor + j) − 1]&1
14:            end for
15:                                          ▷ Finding count
16:            for j = 1 : RowPtr[i + 1] − RowPtr[i] do
17:                if (Bitvec[*(Bitxor + j) − 1]&1) == 0 && Exorall == 1) then
18:                    Bitvec[*(Bitxor + j) − 1] = Bitvec[*(Bitxor + j) − 1] + 2
19:                else if (Bitvec[*(Bitxor + j) − 1]&1) == 1 && Exorall == 0) then
20:                    Bitvec[*(Bitxor + j) − 1] = Bitvec[*(Bitxor + j) − 1] + 2
21:                else if (Bitvec[*(Bitxor + j) − 1]&1) == 0 && Exorall == 0) then
22:                    Bitvec[*(Bitxor + j) − 1] = Bitvec[*(Bitxor + j) − 1] − 2
23:                else if (Bitvec[*(Bitxor + j) − 1]&1) == 1 && Exorall == 1) then
24:                    Bitvec[*(Bitxor + j) − 1] = Bitvec[*(Bitxor + j) − 1] − 2
25:                end if
26:                ▷ Updating the bits and initializing count
27:            end for
28:            for i = 0 : n − 1 do
29:                if (Bitvec[i] & 0Xfe) == 0 then
30:                    Bitvec[i] = Bitvec[i]
31:                else if (Bitvec[i] & 0X80) == 0 then
32:                    Bitvec[i] = 0X01
33:                else
34:                    Bitvec[i] = 0X00
35:                end if
36:            end for
37:        end for
38:        itr = itr + 1
39:    end while
40: end function
```

the received bits and put them in bit nodes as depicted in Figure 8. Then checks are performed in the transferred bits as depicted in Figure 9. Then return bits decide the update value of a bit node by majority as depicted in Figure 10. Finally, the

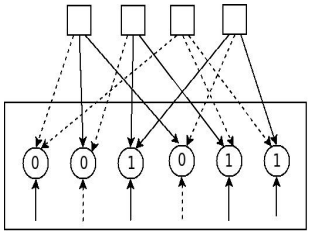

Fig. 10. Bit Update

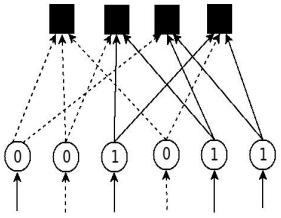

Fig. 11. Test

relation between parity check matrix and code word is checked in Figure 11. If relation satisfies decoding stops else we have to repeat the algorithm again. The Algorithm 2 is pseudo code for the C implementation of the serial decoder.

V. Conclusion & Future Work

Partial parallel implementation is effective as the study on partitioning shows that performance index for all the cases is close to 0.8. In future, the performance of one check serial decoder will be evaluated. Then code will be parallelized at software level. After confirmation of good algorithmic implementation, the code will be converted into its hardware description using Ahir tool chain [10]. This hardware description will be mapped to FPGA target. After successful implementation of serial decoder partial parallel decoder will be designed.

Acknowledgement

I would like to express my gratitude to my guide Prof. Madhav P. Desai, IIT Bombay, whose invaluable guidance, constant encouragement and motivation has inspired me a lot.

References

[1] R. G. Gallager, Low-Density Parity-Check Codes. Cambridge, MA: MIT Press, 1963.

[2] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," IEEE Trans. Inform. Theory, vol. 45, no. 2, pp. 399-431, March 1999.

[3] S.-Y. Chung, G. D. Forney, Jr., T. J. Richardson, and R. L. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," IEEE Commun. Letters, vol. 5, no. 2, pp. 58-60, February 2001.

[4] C. M. Huang, J. F. Huang and C. C. Yang, "Construction of quasi-cyclic LDPC codes from quadratic congruences," in IEEE Communications Letters, vol. 12, no. 4, pp. 313-315, April 2008.

[5] Muhammad Awais and Carlo Condo, "Flexible LDPC Decoder Architectures," VLSI Design, vol. 2012, Article ID 730835, 16 pages, 2012.

[6] Jong-Yeol and H.-J. Ryu, "A 1-gb/s flexible ldpc decoder supporting multiple code rates and block lengths," Consumer Electronics, IEEE Transactions on, vol. 54, pp. 417-424, May 2008.

[7] http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/manual.pdf

[8] C. M. Huang, J. F. Huang, and C. Chin Yang "Construction of quasi-cyclic LDPC codes from quadratic congruences," IEEE Commun. Lett., vol. 12, pp. 313-315, Apr 2008.

[9] http://www.cs.utoronto.ca/ radford/ldpc.software.html

[10] https://github.com/madhavPdesai/ahir/

[11] http://sigpromu.org/sarah/SJohnsonLDPCintro.pdf