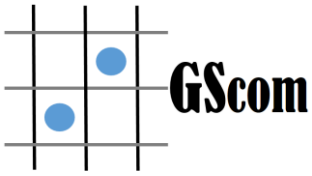

LDPC Algorithm & System Design

Low Density Parity Check (LDPC) Code

Table of Contents

1. Introduction

Low density parity check (LDPC) codes have received major attention for high speed applications in recent years because of their excellent error correction capability and performance. Several architectures have been proposed for LDPC coder/decoder (codec).

GScom has developed an LDPC codec which provides a high throughput with excellent performance in terms of the simplicity and convergence speed by utilizing an irregular block-structured code, optimal degree distribution design process in the irregular block structured parity check matrix, and Parallel Turbo Decoding Message Passing (P-TDMP) Algorithm

2. LDPC System Description

Low-Density-Parity-Check (LDPC)is the one of the most powerful linear block codes in terms of performance (coding gain) and its decoding algorithm is inherently parallel which is attractive for high-speed applications.

In LDPC, we have two types of message passing algorithms- Two Phase Message Passing (TPMP) and Turbo-Decoding Message Passing (TDMP) algorithms. The TDMP has two advantages over the TPMP algorithm: 1. The updated messages are used directly within an iteration to compute new messages and then the refined estimates will spread faster among neighboring nodes which speeds up the convergence. 2. New check messages become directly variable messages during an iteration, hence both the variable and check messages collapses into a single type of messages leading to significant memory savings.

The TDMP algorithm processes the rows of parity check matrix (H) sequentially which results in a lower processing throughput compared to the LDPC-like schedule. The TDMP algorithm can be parallelized by embedding some structure into the parity check matrix, H, that allows parallel processing of multiple rows without communicating messages between the nodes corresponding to these rows. The resulting H is composed of several bands of non-overlapping rows. We refer to the LDPC code with such a structure as an Architecture-Aware LDPC (AA-LDPC) code, and the parallel TDMP as a P-TDMP algorithm.

The parallel Turbo-decoding message-passing (P-TDMP) algorithm was chosen for the AA-LDPC codes due to its faster convergence speed and corresponding throughput advantage over the standard LDPC codes and the standard TDMP codes. The modem employs a reduced complexity message computation mechanism (Soft-Input-Soft-Output (SISO) decoder), which is free of lookup tables and features a programmable network for message interleaving based on the code structure.

To design finite length LDPC codes that can be efficiently implemented on the target multi-processor architecture for high speed communication, we utilize a design process that combines the optimal degree distribution for asymptotic performance, characteristics of structured sub-matrices, finite length code optimization criteria, architectural constraints such as the number of processors, and the width of the SIMD unit. The design process is performed in two levels- the block matrix level which constructs the H_b matrix, and the sub-matrix level which assigns the shift values to the sub-matrices in the H_b block matrix.

2.1 LDPC Encoder

We use AA-LDPC code with the parity check matrix which has a D x B array of S x S sub-matrices. The (n,k) LDPC code has k information bits, n coded bits, and (n-k) parity bits with a code rate of r=k/n. The parity check matrix H is of dimension (n-k) x n, which defines a set of equations as follows:

$$\mathrm{H}\,.\,v^T = 0 \qquad (1)$$

Denote H = $[H_1\ H_2]$, where H_1 and H_2 have dimensions (n-k) x k and (n-k)x(n-k), respectively.

Note that H_1 and H_2 can also be represented as m x b array of SxS sub-matrices and m x m of SxS sub-matrices, respectively, where m = (n-k)/S and b = k/S.

The following Figure 1 shows an example of a block structured parity check matrix H which has D x B array of the SxS sub-matrices, and each sub-matrices is either a zero or a shifted identity matrix with random shift value. Note that D is 4 in Figure 1.

B = n/S

H_1 H_2

I_0		I_5		h_0	I_0		
I_{22}	I_0		I_{12}	h_1	I_0	I_0	
I_7		I_8		h_2		I_0	I_0
I_2	I_{16}			h_3			I_0

Where, h_i is SxS shifted identity matrix and I_x is an identity sub-matrix right shifted by x.

Figure 1. Block-structured irregular matrix

Denote the codeword $\mathrm{v} = [\mathrm{s}\ \mathrm{p}]$, where s is the k information bits and p is the n-k parity bits. Using the decomposed check matrix H and the decomposed codeword v in the above check matrix equation, we have

$$H_1.s^T + H_2.p^T = 0 \pmod 2 \qquad (2)$$

$$p^T = H_2^{-1}H_1.s^T \pmod 2 \qquad (3)$$

Quasi cyclic architecture aware (QC AA) LDPC code provides efficient encoding and decoding with excellent performance. For the parity check matrix in the QC AA-LDPC code, H_2 has a simple deterministic structure where the encoding can be performed recursively using the structure of the QC AA-LDPC code.

The deterministic structure of H_2 enables the encoding procedure to be recursively done. The first column of the sub-matrix in h = [$h_0, h_1, \ldots, h_{m-1}$]^T satisfies $\sum_{i=0}^{c-1} h_i = I_{sxs}$ (mod2). The other column in H_2 has two identity sub-matrices.

As discussed before, H_1 consists of an m x b array of S x S sub-matrices, which are either zero or shifted identity matrices. Given a block of information bits s, if we decompose the information bits into a 1 x b array of 1 x S sub-matrices, and also decompose the p parity bits into a 1 x m array of 1 x S sub-matrices, we have the following recursive parity check vector equation using the decomposed (H, v) in equation (2) above.

$$p_0^T = \sum_{i=0}^{m-1} H_1^{(row\ i)} . s^T \text{ (mod2)}$$

$$p_1^T = H_1^{(row\ 0)} . s^T + h_0 . p_0^T \text{ (mod2)}$$

$$p_2^T = H_1^{(row\ 1)} . s^T + h_1 . p_0^T + p_1^T \text{ (mod2)}$$

……

$$p_{m-1}^T = H_1^{(row\ m-2)} . s^T + h_{m-2} . p_0^T + p_{m-2}^T \text{ (mod2)}$$

Since H_1 consists of either zero or shifted identity sub-matrices, the $H_1^{(row\ i)} s^T$ can be efficiently implemented by a S-bit barrel shifter, a S-bit XOR and a S-bit Register, as shown in the following Figure 2. With all the $H_1^{(row\ i)} s^T$ determined, the parity bit vectors p_0 through p_{m-1} can be found recursively.

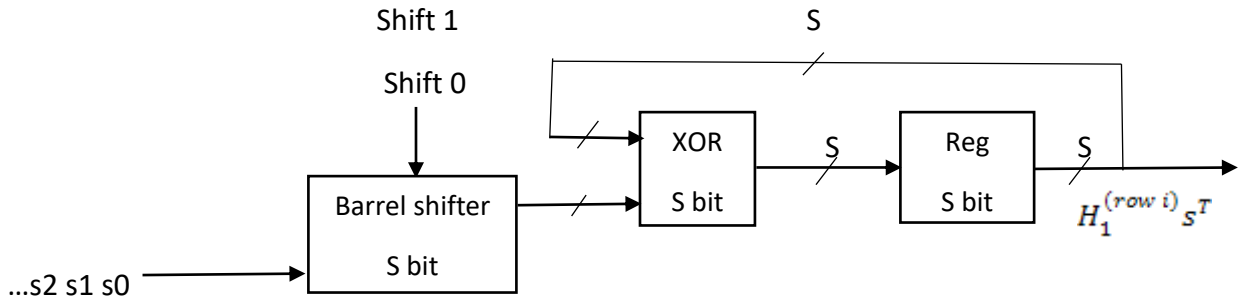

Figure 2. Circuit to calculate $H_1^{(row\ i)} s^T$

2.2. LDPC Decoder Description

The classic LDPC decoding algorithm is a two-phase message passing (TPMP) algorithm or the so called belief propagation (BP) algorithm. The decoding procedure revolves around the two-phase transmission of extrinsic information between check nodes and bit nodes on the Tanner Graph. The message passing decoding algorithm utilizes the bit-to-bit dependence required in check-sum equations of the LDPC code to adjust the probability of each bit until obtaining a valid code-word.

The Turbo Decoding Message Passing (TDMP) algorithm outperforms the standard two-phase decoding algorithm with its faster convergence speed of roughly a factor of two, in terms of the number of decoding iterations required, and with its memory savings of more than 50%. With the TDMP scheme both variable and check messages collapse into a single type of message, and compared to the TPMP algorithm the TDMP algorithm requires lower complexity and is therefore more suitable for hardware implementation.

The Quasi Cyclic LDPC (QC-LDPC), used in our system, is an Architecture Aware LDPC (AA-LDPC). The ones in each block row (every s row) in the QC-LDPC are not overlapped. As a result, decoding can be processed for s rows simultaneously. The TDMP algorithm can be modified to a parallel version referred to as P-TDMP using the architecture of the AA-LDPC. The P-TDMP algorithm provides an improvement in decoding throughput over the ordinary TDMP algorithm and is attractive for high speed applications.

One desirable characteristic of the AA of parity-check matrix would be to divide the variable nodes and check nodes into the clusters of size S such that if one node from a cluster connects to a node from another cluster, then there exist distinct connections between all nodes from both clusters. The connections between the nodes of two clusters are done according to some permutation π, where π varies between cluster pairs.

In the LDPC we used, the parity matrix H is decomposed into SxS binary sub-matrices. These sub-matrices satisfy the following two conditions- each column has at most a single 1, and each row has at most a single 1. Note that zero columns, zero rows and null matrices are allowed under this definition.

By decomposing the parity check matrix H of an LDPC code in such a way as to restrict the column positions of the ones, the LDPC decoding problem is transformed into a turbo-decoding problem where messages flow in tandem only between the adjacent super-codes as opposed to potentially all the sub-codes on the parity check matrix. The inter-leavers are factored into smaller inter-leavers that are more practical to implement.

2.2.1. LDPC Soft Decoding Algorithm

We review the LDPC decoding algorithm using a classical two phase message passing (TPMP) approach, called iterative layered belief propagation approach, using the following parity check matrix as an example.

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

In the above parity check matrix, we have four check nodes (m = 4) and eight variable nodes (j=8). Let $L(q_{mj})$ denote the variable node log likelihood ratio (LLR) message sent from variable node j to the check node m, then:

$$L(q_{mj}) = L(q_j) - R_{mj},$$

$$R_{mj} = \prod_{j' \in N(m)\setminus\{j\}} sign(L(q_{mj'}))\ \Psi\left[\sum_{j' \in N(m)\setminus\{j\}} \Psi(L(q_{mj'}))\right]$$

$$L(q_j) = L(q_{mj}) + R_{mj},$$

$$\Psi(x) = -\log\left[\tanh\left(\frac{|x|}{2}\right)\right]$$

Where, R_{mj} is the check node LLR message sent from the check node m to the variable node j, and $L(q_j)$ (j=1,2,…..,N) represents the a posteriori probability ratio (APP) for all variables. The APP messages are initialized with the channel reliability values of the coded bits. N(m) is the set of all variable nodes connected to the check node m.

The LDPC is an iterative decoding scheme which performs the above three parameter estimations and updates in each iteration step to improve the reliability of the bit estimation using LLR information and parity check equations. If all parity check equations are satisfied or the pre-determined maximum number of iteration is reached, then the decoding stops.

2.2.2. Turbo Decoding Message Passing (TDMP) Algorithm

The previous decoding algorithm operates using two types of messages and requires the saving all intermediate messages between both rounds at every iteration. Moreover, newly computed messages in a round of computations do not participate in further message computations until the decoding iteration is over. If updated messages are used directly within an iteration to compute new messages, then refined estimates will speed up the convergence behavior of the algorithm. Further, new check messages become directly variable messages within the iteration, hence, both variable and check messages collapse into a single type of message leading to a significant savings in the memory storage required.

The TDMP algorithm utilizes the most recently updated message directly within the iteration to compute a new message, which speeds up the convergence speed in terms of total iterations required. Furthermore, the two phase message computation in the TPMP algorithm is substituted by a single

computation, and the memory savings is significant. As a result, the TDMP algorithm outperforms the TPMP algorithm in terms of throughput (20%-50%), coding gain, and hardware efficiency.

The TDMP algorithm can be described with the help of the parity check matrix in the above figure in Section 3.5.2.1 which has four parity check equations corresponding to a code of length 8. The algorithm is based on decoding the rows (parity check equations) of the parity check matrix sequentially. Extrinsic messages generated from decoding earlier rows are used as prior messages to decode subsequent rows. To each row i in H, we associate the vector $\underline{\lambda}^i = [\lambda_1^i, \dots\dots, \lambda_{c_i}^i]$ of extrinsic messages corresponding to the nonzero entries in that row. The number of nonzero c_i in a row is called the weight. Let I_i denote the set of indexes of ones in row I and $\underline{\gamma}(I_i)$ represent the posterior messages of row i. For example, in the parity check matrix H above in the figure shown in Section 3.5.2.1, the weight of row 2 is c_2=4, while its index set I_2={1,4,6,7}, and its extrinsic message vector $\underline{\lambda}^2$=[$\lambda_1^2, \lambda_2^2, \lambda_3^2, \lambda_4^2$]. Where λ_1^2 corresponds to bit 1 and λ_2^2 corresponds to bit 4, etc. The extrinsic messages $\underline{\lambda}^i$ are associated to the extrinsic estimates about the bits from all parity check equations that these bits participate. Let $\underline{\gamma}$= $[\gamma_1, \dots\dots, \gamma_N]$ denote posteriori messages that stores the sum of all messages generated by the rows in which each bit participates. For example, N=8, $\gamma_1=\lambda_1^2 + \lambda_1^3$, $\gamma_2=\lambda_1^1 + \lambda_2^3$,…,$\gamma_8=\lambda_4^1 + \lambda_4^3$. The posterior messages of row i are indexed as $\underline{\gamma}(I_i)$ and the hard decisions are determined by slicing the $\underline{\gamma}$ vector.

Decoding the *i* th parity check row involves the following four steps which constitute a decoding sub-iteration:

1) Read: $\underline{\lambda}^i$ and $\underline{\gamma}(I_i)$ are read for row i from memory.
2) Subtract: $\underline{\lambda}^i$ are subtracted from $\underline{\gamma}(I_i)$ to generate prior messages $\underline{\rho}$ = $[\rho_1, \dots\dots, \rho_{c_i}]$.
3) Decode: Decode row i using a SISO algorithm with $\underline{\rho}$ as input and $\underline{\Lambda}^i$ = $[\Lambda_1^i, \dots\dots, \Lambda_{c_i}^i]$ as output.
4) Write back: Replace the original extrinsic message $\underline{\lambda}^i$ with $\underline{\Lambda}^i$ and update $\underline{\gamma}(I_i)$ by $\underline{\gamma}(I_i)$ = $\underline{\rho}$ + $\underline{\lambda}^i$.

A decoding iteration comprises multiple sub-iterations corresponding to the rows of H. A round of sub-iterations over all rows of H constitutes a decoding iteration. The whole decoding procedure will be terminated when it satisfies all the parity check equations in the matrix or the number of iterations has reached a predefined number. The TDMP algorithm requires storage memory for

$$M_{TDMP} = \sum_{i=1}^{M} c_i \; + \; N$$

messages, assuming H has M rows, N columns, and row weights c_i.

A formal description of the TDMP algorithm is given using the SISO algorithm which is a reduced-complexity message computation algorithm. The SISO algorithm will be explained in detail in the next section. The inputs to the TDMP algorithm are a sparse parity check matrix $H_{M\,X\,N}$ representing a repeated accumulated (RA) code, LDPC code (where N=n+k), input channel observations $\underline{\delta}$ = $[\delta_{u_1}, \dots\dots, \delta_{u_k}; \delta_{y_1}, \dots\dots, \delta_{y_n}]$, and maximum number of decoding iterations T. The outputs are hard decision estimates of the information bits $\underline{u} = [u_1, u_2, \dots\dots, u_k]$.

The pseudo-code of the TDMP algorithm is listed below. The storage for the extrinsic message and posterior message is allocated using the vectors $\underline{\lambda^i}$, $i = 1,2, \ldots \ldots, M$, and the vector $\underline{\gamma}$, respectively. The main body of the algorithm has two nested loops. The outer loop runs until the desired number of iteration is reached, while the inner loop runs over the rows of $H_{M\,X\,N}$ sequentially from 1 to M. As described in the four steps above, for every row *i,* the posterior messages of its nonzero entries $\underline{\gamma}(I_i)$ as well as the extrinsic messages $\underline{\lambda^i}$ are read (1st step). Next, the extrinsic messages $\underline{\lambda^i}$ are subtracted from $\underline{\gamma}(I_i)$ and stored in a temporary vector $\underline{\rho}$ (step 2). The messages $\underline{\rho}$ are then passed to a SISO decoder which generates the updated messages $\underline{\Lambda}$ (step 3). Finally the extrinsic messages $\underline{\lambda^i}$ are updated with the newly updated messages $\underline{\Lambda}$, and the posterior messages are updated with the new sum $\underline{\gamma}(I_i) = \underline{\rho} + \underline{\Lambda}$ (step 4).

TDMP Algorithm:

//Storage: M extrinsic message vectors $\underline{\lambda^i} = [\lambda_1^i, \lambda_2^i, \ldots \ldots, \lambda_{c_i}^i]$ of length c_i, $i = 1,2, \ldots ., M$.

//Storage: Vector of N posterior messages $\underline{\gamma} = [\gamma_1, \ldots \ldots, \gamma_N]$.

//Initialization

$\underline{\lambda^i} \longleftarrow \underline{0}$, $i = 1,2, \ldots . . M$

$\underline{\gamma} \longleftarrow \underline{\delta}$

//Iterative decoding

for t = 1 to T do

    for i = 1 to M do

        $\underline{\rho} \longleftarrow \underline{\gamma}(I_i) - \underline{\lambda^i}$ //Read and subtract

        $\underline{\Lambda} \longleftarrow$ SISO($\underline{\rho}$) // Decode

        $\underline{\lambda^i} \longleftarrow \underline{\Lambda}$ //Write back

        $\underline{\gamma}(I_i) \longleftarrow \underline{\rho} + \underline{\Lambda}$ //Write back

    end for

end for

//Hard decision

$u_j \longleftarrow \frac{1}{2}(sgn(\gamma_j) + 1)$, $j = 1, \ldots \ldots, k$

2.2.2.1. SISO algorithm for message decoding in TDMP algorithm

The SISO (soft input soft output) algorithm is adopted for message decoding in the TDMP algorithm as can be seen in the decoding step in its sub-iteration procedure. Unlike other scheme of

$$\Lambda_j^i = \Psi^{-1}\left(\sum_{l\epsilon l\backslash j}\Psi(\rho_l)\right), \qquad i = 1, \ldots., M;\ \ j = 1, \ldots., c_i$$

the SISO decoder need not use a lookup table, and can be implemented using simple logic gates. Moreover, the SISO circuit can be implemented using the "max-quartet" function $\mathrm{Q(x,y)}$ as can be seen in Figure 11 resulting in memory savings, an improvement in efficiency for the hardware implementation, and it provides a more accurate approximation than any other available at this time.

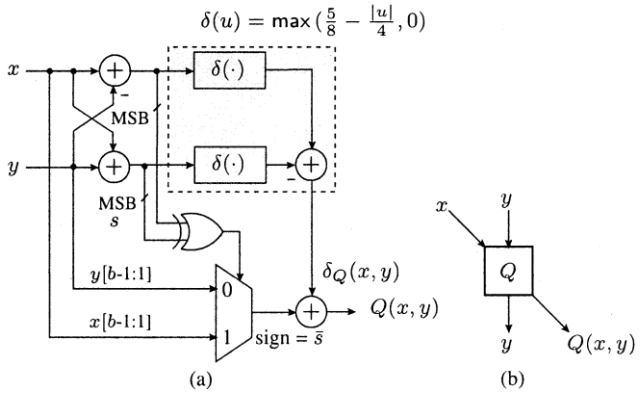

Figure 3. Max-quartet function $\mathrm{Q(x,y)}$: (a) Logic circuit, and (b) Symbol

The pseudo-code of the SISO algorithm is listed below.

$\Lambda = \mathrm{SISO}(\rho)$

// Input: $\underline{\rho} = [\rho_1, \rho_2, \ldots., \rho_c]$

// Output: $\underline{\Lambda} = [\Lambda_1, \Lambda_2, \ldots\ldots, \Lambda_c]$

// Initializations

$\alpha_1 \leftarrow \rho_1$

$\beta_c \leftarrow \rho_c$

// Forward-backward recursion

for i = 1 to c-2 do

j = c – (i-1)

$\alpha_{i+1} \leftarrow Q(\alpha_i, \rho_{i+1})$

$\beta_{j-1} \leftarrow Q(\rho_{j-1}, \beta_j)$

if $c/2 \leq i \leq c-2$, then

$\Lambda_{i+1} \leftarrow Q(\alpha_i, \beta_{i+2})$

$\Lambda_{j-1} \leftarrow Q(\alpha_{j-2}, \beta_j)$

end if

end for

$\Lambda_1 \leftarrow \beta_2$

$\Lambda_c \leftarrow \alpha_{c-1}$

Where

$$Q(x,y) = \max(x,y) + \max\left(\frac{5}{8} - \frac{|x-y|}{4}, 0\right) - \max(x+y, 0) - \max\left(\frac{5}{8} - \frac{|x+y|}{4}, 0\right)$$

The SISO message computation process starts with the initialization of α_1 and β_c with ρ_1 and ρ_c, respectively, and then the forward and backward recursion can be handled at the same time. For the forward recursion, $Q(\rho_1,\rho_2) = \alpha_2$, then $Q(Q(\rho_1,\rho_2),\rho_3) = Q(\alpha_2,\rho_3) = \alpha_3$,……., until $Q(\alpha_{j-2},\rho_{j-1}) = \alpha_{j-1}$. For the backward recursion, $Q(\rho_{n-1},\rho_n) = \beta_{n-1}$, then $Q(\rho_{n-2}, Q(\rho_{n-1},\rho_n)) = Q(\rho_{n-2},\beta_{n-1}) = \beta_{n-2}$,…… up to $Q(\rho_{j+1},\beta_{j+2}) = \beta_{j+1}$. When α_{j-1} and β_{j+1} are known, the first output $\Lambda_j = Q(\alpha_{j-1},\beta_{j+1})$ can be calculated. Then, with the forward and backward recursion updating, the output messages can be carried out in pairs. Finally, β_2 and α_{c-1} are assigned to the leftmost output Λ_1 and the rightmost output Λ_c, respectively. We can illustrate the SISO algorithm assuming c is 6 and the input of $\underline{\lambda}$ in the following Figure 4.

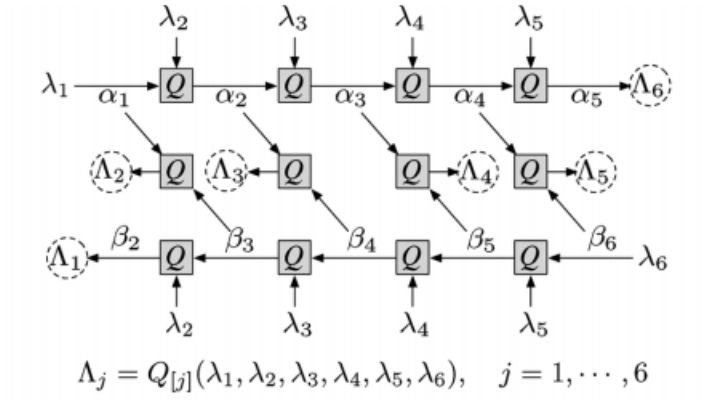

Figure 4. SISO Algorithm using max-quartet function

2.2.3. Parallel Turbo Decoding Message Passing (P-TDMP) Algorithm

The TDMP algorithm processes the rows of H sequentially which results in a lower processing throughput compared to the LDPC-like schedule. The TDMP algorithm can be parallelized by embedding some structure into the parity check matrix, H, that allows parallel processing of multiple rows without communicating messages between the nodes corresponding to these rows. The resulting H is composed of several bands of non-overlapping rows. We refer to the LDPC code with such a structure as an architecture-aware LDPC (AA-LDPC) code, and the parallel TDMP as a P-TDMP algorithm.

In the architecture-aware LDPC parity check matrix H, the ones in each block row are not overlapped. Therefore, decoding can be processed for s rows simultaneously at every iteration cycle. So, the TDMP algorithm in the AA-LDPC code can be modified into a parallel version of the TDMP, named as P-TDMP. The P-TDMP provides an improvement in decoding throughput over the conventional TDMP algorithm.

The P-TDMP algorithm is based on having the ones in every s number of rows of sub-matrices as non-overlapped, so the decoding procedure can be handled in parallel using s SISO decoders. The s SISO decoders constitute a decoder group working simultaneously. Decoder i processes row i in each sub-matrix and maintains the extrinsic messages $\lambda^{i,j}$ in local memory, while posteriori messages are passed to the decoders s messages at a time from a global memory. The decoding procedure runs over the rows of H in c iterations and processes s rows simultaneously for every iteration which improves the de-coding throughput by a factor of s as compared to the TDMP algorithm. The AA-TDMP reduces the required number of multiplexers/de-multiplexers by an order of n (O(n)), and the required number of control overhead by an order of O(log n).

The parallel Turbo-decoding message-passing (P-TDMP) algorithm is chosen for the AA-LDPC codes providing a faster convergence speed and hence a throughput advantage over the standard TDMP codes. We employ a reduced complexity message computation mechanism (SISO algorithm) and the features of a programmable network for message interleaving, based on the code structure.

To design finite length LDPC codes that can be efficiently implemented on the target multi-processor architecture for high speed communication, we utilize a design process that combines the optimal degree distribution for asymptotic performance, characteristics of structured sub-matrices, finite length code optimization criteria, architectural constraints such as the number of processors, and the width of the SIMD unit. The design process is performed at two levels. First is the block matrix level design which constructs the H_b matrix, and second is the sub-matrix level design which assigns shift values to the sub-matrices.

The S rows of ones in each row of sub-matrices in an AA-LDPC H are non-overlapping, and hence can be processed in parallel using S SISO decoders. Decoder s processes row s in each row of sub-matrices, for a total of D rows, and maintains the extrinsic messages denoted by $\underline{\lambda}^{d,s},\ \mathrm{d} = 1,2,\ldots.,\mathrm{D},$ in a local memory. Posterior messages are stored in a global memory of size N and passed in parallel (S messages at a time) to the decoders using a network that implements the factored edge permute process.

The pseudo-code of the P-TDMP algorithm is listed below. The innermost parallel loop runs over the rows of H in D iterations processing S rows during each iteration period. This results in a factor of S improvement in decoding throughput over the TDMP algorithm.

P-TDMP Algorithm: $\underline{u} = P - TDMP(H, \underline{\delta}, D, S, T)$

//Input: AA-parity check matrix H

//Input: Channel value $\underline{\delta} = [\delta_{u_1}, \ldots \ldots, \delta_{u_k}; \delta_{y_1}, \ldots \ldots, \delta_{y_n}]$

//Input: D = number of sub-matrices contained in each block column of H

//Input: S = dimension of sub-matrices

//Input: T = max number of iterations

//Output: $\underline{u}$ = hard decision estimates vector

//Storage: D x S extrinsic message vectors $\underline{\lambda}^{d,s} = [\lambda_1^{d,s}, \lambda_2^{d,s}, \ldots \ldots, \lambda_{c_{d,s}}^{d,s}]$

of length $c_{d,s}$, $d = 1, \ldots \ldots, D$, $s = 1, \ldots, S$

//Storage: Vector of N posterior messages, $\underline{\gamma} = [\gamma_1, \ldots \ldots, \gamma_N]$

//Initialization

$\underline{\lambda}^{d,s} \longleftarrow \underline{0}$, $d = 1, \ldots \ldots, D$, $s = 1, \ldots, S$

$\underline{\gamma} \longleftarrow \underline{\delta}$

//Iterative decoding

for t = 1 to T do

for d = 1 to D do for all s

$\underline{\rho}^s \longleftarrow \underline{\gamma}(I_{(d-1)S+s}) - \underline{\lambda}^{d,s}$ //Read and subtract

$\underline{\Lambda}^s \longleftarrow$ SISO($\underline{\rho}^s$) // Decode

$\underline{\lambda}^{d,s} \longleftarrow \underline{\Lambda}^s$ //Write back

$\underline{\gamma}(I_{(d-1)S+s}) \longleftarrow \underline{\rho}^s + \underline{\Lambda}^s$ //Write back

end for

end for

//Hard decisions

$u_j \longleftarrow \frac{1}{2}(sgn(\gamma_j) + 1)$, $j = 1, \ldots \ldots, k$

2.2.3.1. Programmable P-TDMP Decoder Architecture

Figure 5 shows one architecture option for a high level programmable decoder implementing the P-TDMP algorithm. We assume that the corresponding parity check matrix is for an AA-LDPC code and has the following parameters:

1) S is the size of the sub-matrix partitions (assuming permutation matrices).

2) B = n/S denotes the number of sub-matrices per block row.

3) D = m/S denotes the number of sub-matrices per block column.

4) r is the maximum number of permutation matrices per block row.

5) $c_1, \ldots \ldots, c_B$, corresponds to the number of permutation matrices in the block columns.

The architecture includes B memory modules for storing the messages, S MPUs (message processing units) that operate in parallel, and read/write networks for transporting the messages between the memory and the MPUs. The parameter S acts as a scaling parameter.

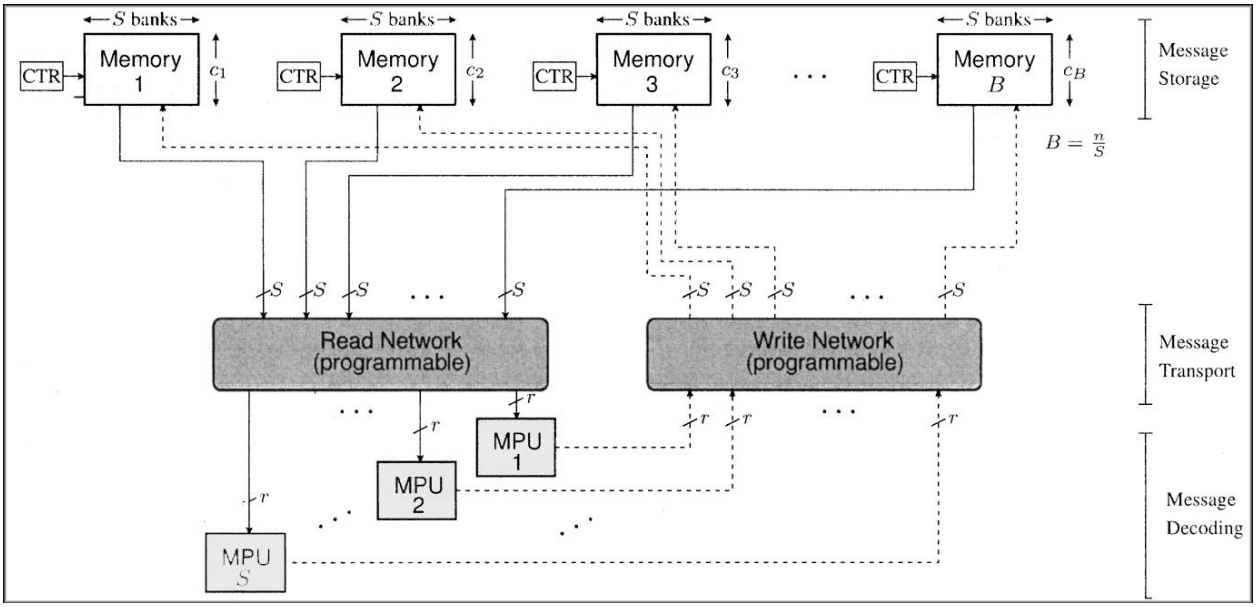

Figure 5. Decoder architecture implementing the P-TDMP algorithm

The decoder completes a single decoding iteration by performing a round of D updates across the super-codes. An update corresponding to a single super-code C^j constitutes a sub-iteration which involves the following four steps:

1) The read-network performs c read operations from memory, where c is the maximum node degree of C^j. It then forwards r messages to each of the S MPUs.
2) The MPUs update the messages in parallel with SISO algorithm.
3) The updated messages are routed by the write-network to the appropriate memory modules scheduled to receive messages related to C^j.
4) The messages are written in the designated memory modules and the address counters are updated.