Low Latency Algorithms for Iterative Codes

Seok Soon Choi* ***Associate Member***, Ji Won Jung* ***Regular Member***,
Jong Tae Bae*, Min Hyuk Kim*, ***Associate Members***, Eun A Choi** ***Regular Member***

ABSTRACT

This paper presents low latency and/or computation algorithms of iterative codes of turbo codes, turbo product codes and low density parity check codes for use in wireless broadband communication systems. Due to high coding complexity of iterative codes, this paper focus on lower complexity and/or latency algorithms that are easily implementable in hardware and further accelerate the decoding speed.

Key Words : Iterative Codes, Turbo Code, Low Density Parity Check Code, Turbo Product Code, Low Latency

Ⅰ. Introduction

Iterative decoding based on symbol-by-symbol soft-in/soft-out decoding algorithm has significant attention, due to its near Shannon-limit error performance for decoding of turbo codes [1], low density parity-check (LDPC) code [2] and turbo product code [3]. Like maximum a posterior probability (MAP) decoding, iterative decoder processes the received symbols recursively to improve the reliability of each symbol based on constrains that specify the code. In the first iteration, the decoder only uses the channel output, and generates soft output for each symbol. The output reliability measures of the decoded symbols at the end of each decoding iteration are used as input for next iteration. Therefore, the latency and complexity caused by several iterations and high computation order, it can be difficult to implement the decoding in hardware and to apply the high-speed wireless applications [4]. To solve the latency problem, early-stop algorithms that the decoding iteration processes until a certain stopping condition is satisfied, can be applied. Furthermore, fully parallel decoder structure also can be applied for iterative decoders to reduce the latency. Therefore, early-stop algorithm and parallel fashioned decoder can be applied to all of the iterative codes in common. However, the standard LDPC codes can have large codeword length and it can be difficult to implement hardware in a fully parallel way. For example, The Digital Video Broadcasting (DVB-S2) recommends that LDPC coded block size be 64800, and the number of iteration be about 70 in the case of half coding rate. A large number of iterations for a large block size gives rise to a large number of computation operations, mass power consumption, and decoding delay. Furthermore, a large number of block size make impossible to implement by fully parallel way. It is necessary to use partial parallel way. In [5], "shuffled" method were presented to reduced the required number of iterations. However it requires double hardware size compared to conventional one. In LDPC codes cases, this paper proposes two kinds of simplified complexity-reduced algorithm. First, sequential decoding with partial group is proposed. It has the same H/W complexity, and a few numbers of iteration are required at the same performance in comparison with a conventional decoder algorithm. Secondly, an early detection method for reducing the computational complexity is proposed. Using a confidence criterion, some bit

* 한국해양대학교 전파공학과 위성통신연구실 (ms43bjt@hhu.ac.kr), ** 한국전자통신연구원 광대역 무선 멀티미디어 연구팀
논문번호 : KICS2006-03-145, 접수일자 : 2006년 3월 28일, 최종논문접수일자 : 2006년 12월 20일

nodes and check node edges are detected early on during decoding. In this way, because early detected edges are not computed from following iterations, the computational complexity of further processing is reduced. In turbo codes cases, the latency caused by serially calculations of forward and backward metric can be dramatically reduced by using a radix-4 and dual-path processing.

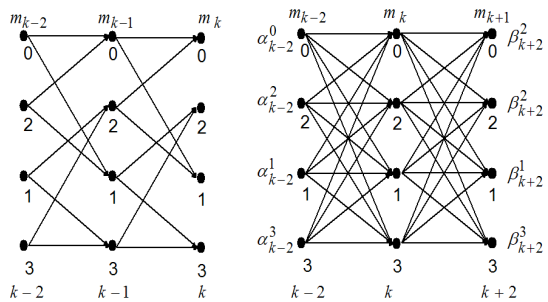
The aim of this paper is to introduce several low-latency and/or complexity algorithms which further accelerate the decoding speed for turbo codes, LDPC codes and turbo product codes.

Ⅱ. Low Latency Algorithms of Turbo Code

Since convolutional turbo codes are very flexible codes, easily adaptable to a large rate of block sizes and coding rates, they have been adopted in the DVB standard for Return Channel via Satellite(DVB-RCS). The use of RCST(RCS Terminal) includes individual and collective installation(e.g. SMATV) in domestic environment. However, the applications of turbo codes are limited to specific data such as low data-rate services because of their limit of decoding speed. Therefore, it is highly required to develop the high-speed turbo decoder. To solve the problem with latency of turbo decoder, four kinds of algorithms are introduced. The first algorithm is radix-4 algorithm and the second algorithm is the dual-path processing algorithm. The third algorithm is the full parallel decoding algorithm. The fourth algorithm is the early-stop algorithm based on hard-decision-aided (HDA) scheme. The decoding iteration processes until a certain stopping condition is satisfied Then hard decisions are made based on the reliability measures of the decoded symbol at the last decoding iteration

2.1 Radix-4 Algorithm

The first algorithm is the radix-4 decoding algorithm, where the previous state at $t=k$-2 goes forward to the current state at $t=k$, and the reverse state at $t=k$+2 goes backwards from the current one such that the time interval from $t=k$-2 to $t=k$ is merged into $t=k$. Therefore, we can decode two source data bits at the same time without any performance degradation while reducing the block size buffered in memory. Using the unified approach to state metrics, a 2^{v-1} -state trellis can be iterated from time index n-k to n by decomposing the trellis into 2^{v-k} sub-trellises, each consisting of k iterations of a 2^k -state trellis. Each 2^k -state's sub-trellis can be collapsed into an equivalent one-stage radix-2^k trellis by applying k levels of look-ahead to the recursive update. Collapsing the trellis does not affect the decoder performance since there is a one-to-one mapping between the collapsed trellis and radix-2 trellis. An example of the decomposition of a 4-state radix-2 into an equivalent radix-4 trellis using one stage of look-ahead is shown in Fig. 1, where v=3, $g_1=(7)_{octal}$, $g_2=(5)_{octal}$ with v denoting the constraint length.



(a) 4-state radix-2 trellis (b) 4-state radix-4 trellis

Fig. 1. Four-state radix-2 to radix-4 trellis

2.2 Dual-Path Processing Algorithm

In a conventional scheme, the decoder must wait for finishing the backward state metric (BSM) (or forward state metric (FSM)) calculations before calculating the extrinsic information. The dual-path processing method doesn't need to wait. The decoder calculates the FSM (left to right) and BSM (right to left), simultaneously. When the FSM and BSM reach the same point, then the decoder begins to calculate the extrinsic information. Fig. 2 shows the operation of dual-path processing.

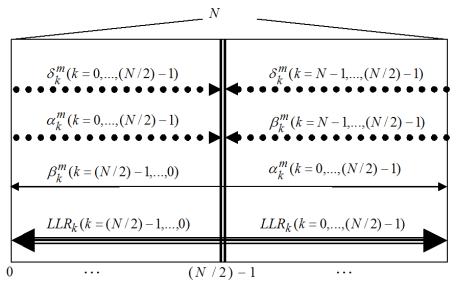


Fig. 2. Dual-path processing algorithm

The procedure of the dual-path processing is as follows.

Step 1: Initialize the forward state metric and backward state metric.

$$\begin{aligned} \alpha_0^i(s_0^i(m)) &= 1 \quad for \quad m=0 \\ &= 0, \quad else \\ \beta_N^i(s_b^i(m)) &= 1 \quad for \quad m=0 \\ &= 0, \quad else \end{aligned} \tag{1}$$

$\alpha_k^i(m)$ and $\beta_k^i(m)$ are FSM and BSM at time of *k*, information bit of *i*, and state of *m*.

Step 2: After receiving the whole set of received symbols of N, FSMs (left to right) and BSMs (right to left) are calculated simultaneously.

$$\hat{\alpha}_k^i(m) = \exp\left(\frac{2}{\sigma^2}(x_k i + y_k Y_k(i,m))\right) \sum_{j=0}^{1} \hat{\alpha}_{k-1}^j(S_b^j(m)) \quad (k=0,...,(N/2)-1) \tag{2-1}$$

$$\hat{\beta}_k^i(m) = \sum_{j=0}^{1} \hat{\beta}_{k+1}^j(m) \exp\left(\frac{2}{\sigma^2}(x_{k+1} j + y_{k+1} Y_{k+1}(j, S_f^i(m)))\right) \quad (k=(N-1),...,(N/2)) \tag{2-2}$$

Step 3: At the middle point, begin to calculate the log likelihood ratios (LLR).

$$\vec{L(d_k)} = \log \frac{\sum_m \alpha_k^1(m)\beta_k^1(m)}{\sum_m \alpha_k^0(m)\beta_k^0(m)} \quad (k=(N/2),...,(N-1)) \tag{3-1}$$

$$\overleftarrow{L(d_k)} = \log \frac{\sum_m \alpha_k^1(m)\beta_k^1(m)}{\sum_m \alpha_k^0(m)\beta_k^0(m)} \quad (k=(N/2)-1,...,0) \tag{3-2}$$

$\vec{L(d_k)}$ means LLR outputs in the direction of right to left and $\overleftarrow{L(d_k)}$ means LLR outputs in the direction of left to right.

2.3 Parallel Algorithm

Different from the original turbo decoder consisting of two decoders concatenated in a serial fashion, we present a parallel decoder structure using the parallel sum, where the two decoders operate in parallel and update each other immediately and simultaneously after each one has completed its decoding. In decoding the estimated data, we use the sum of the LLR outputs of the parallel decoders to reduce the latency to half while maintaining the same performance level.

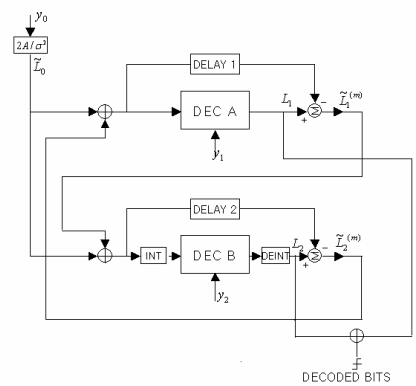


Fig.3. Parallel structure of turbo decoder

2.4 Early Stop Algorithm

The decoding iteration processes until a certain stopping condition is satisfied, hard decisions are made based on the reliability measures of the decoded symbol at the last decoding iteration. HAD algorithm is used as an early-stop algorithm. It compares each decision generated by the two decoders, and when the two sets of decisions match, it stops decoding on the current block and outputs the hard decision bits. Table 1 shows the average

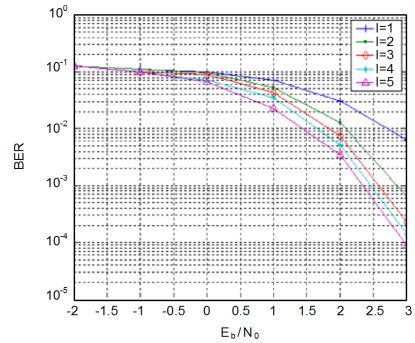
number of iterations in an HAD algorithm. At an E_b/N_0 of 2 dB, it requires about 2.8 iterations relative to 8 for a given performance. This means that the decoding speed is improved or the power consumption (cost) is reduced by 64.6 %.

Table 1. The average number of iterations according to E_b/N_0 (the predetermined number of iterations is 8)

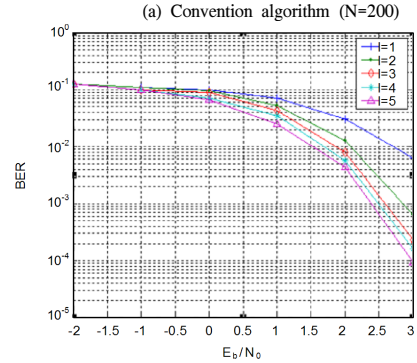
E_b/N_o[dB]	Average number of iterations	Decoding speed improvement (%)
1	6.21	22.4
1.5	4.31	46.1
2	2.83	64.6

2.5 Simulation Results

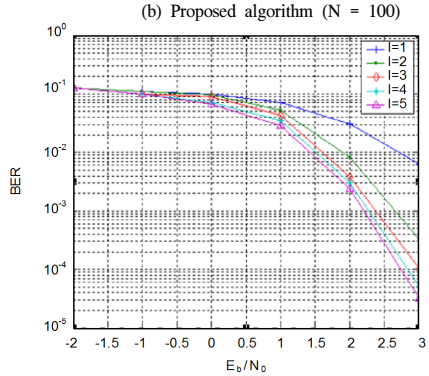
The bit-error rate (BER) performance of the new high-speed turbo decoder architecture combining the four schemes is analyzed in this section. For a comparison purpose, Fig. 4 shows the performance of the new decoder and a conventional one using v=3 turbo codes with generator polynomials $g_1=(7)_{octal}$, $g_2=(5)_{octals}$ a function of interleaving size N and as a function of the number of iterations I. In the radix-4 method, the symbol interleaving, takes an information stream of length N_s composed of 2-bit words and feeds it to a random interleaving. From the figure, it can be verified that the performances of the proposed decoder architecture is very close to the conventional decoder for small block sizes (less than 300 bits). For large interleaving sizes (more than 300 bits), the performance of the new decoder is slightly degraded relative to the conventional one because the randomness of the symbol interleaving is reduced. In addition, although parallel fashioned decoder reduces the decoding delay of serial decoding by half, the extrinsic messages are not taken advantage of as soon as they become available, because the extrinsic messages are delivered to component decoders only after each iteration is complete. This disadvantage is maybe solved by applying "replica shuffled algorithm" in Reference [5].



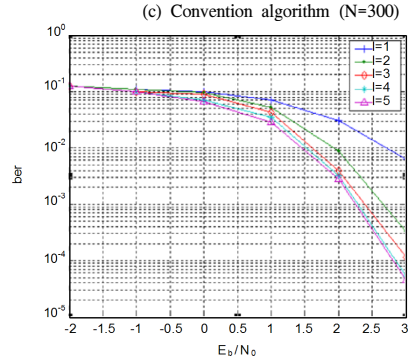
(a) Convention algorithm (N=200)



(b) Proposed algorithm (N = 100)



(c) Convention algorithm (N=300)



(d) Proposed algorithm (N = 150)

Fig. 4. Performance of the proposed decoder over an AWGN channel compared with that of a conventional algorithm as a function of interleaver size and number of iterations.

Ⅲ. Low Latency Algorithms of LDPC codes

The high definition television (HDTV) satellite standard, known as the Digital Video Broadcasting (DVB-S2) transmission system, employs a low density parity check (LDPC) coding technique as a channel coding scheme. Unlike turbo codes, LDPC codes have an easily parallelizable decoding algorithm, which consists of simple operations such as addition, comparison, and creation of a look-up table. Moreover, the degree of parallelism is 'adjustable', which makes it easy to trade-off both throughput and complexity. However, the DVB-S2 system requires a large block size and large number of iterations to near Shannon's limit. The standard recommends that the LDPC coded block size be 64800, and the number of iteration be about 70 in the case of a half-coding rate. A large number of iterations for a large block size give rise to a large number of computation operations, mass power consumption, and decoding delay. It is necessary to reduce the iteration numbers and computation operations without performance degradation in order to implement an LDPC decoder with low power consumption. This paper proposes two kinds of simplified complexity-reduced algorithms. First, sequential decoding with a partial group is proposed. It has the same hardware complexity, and a fewer number of iterations is required at the same level of performance in comparison with a conventional decoder algorithm. The computation of bit node weights and check node weights can seem to be as an approximate projection based on the parity-check matrix; the grouping simply says that this projection can be done in several steps to obtain an approximate answer. Secondly, an early detection method for reducing the computational complexity is proposed. Using a confidence criterion, some bit nodes and check node edges are detected early on during decoding. In this way, because early detected edges are not computed from following iterations, the computational complexity of further processing is reduced.

3.1 LDPC Decoding Algorithm

The purpose of the decoder is to determine the transmitted values of the bits. Bit nodes and check nodes communicate with each other to accomplish this goal. The decoding starts by assigning the channel values to the outgoing edges, from bit nodes to check nodes. Upon receiving them, the check nodes make use of the parity check equations to update the bit node information and send it back. Each bit node then performs a soft majority vote among the information reaching him. At this point, if the hard decisions on the bits satisfy all of the parity check equations, it indicates that a valid codeword has been found and the process stops. Otherwise, the bit nodes go on sending the results of their soft majority votes to the check nodes. In the following sections, we describe the decoding algorithm in detail. The number of edges adjacent to a node is called the degree of that node.

Step 1. Initialization

The decoding starts by assigning the channel transmit values, r_n , to the outgoing edges, from bit nodes to check nodes. The initial channel value is shown in (1).

$$u_n = -L_c \cdot r_n (L_c = \frac{2}{\sigma^2}), \quad n = 0,1,\ldots,N-1 \tag{4}$$

where, N is the codeword size and σ is Gaussian noise variance.

Step 2. Check Node Update

Let us denote the incoming messages to the check node k from its dc adjacent bit nodes by $v_{n_1 \to k}$, $v_{n_2 \to k}$, $\cdots$, $v_{n_{dc} \to k}$, as shown in Fig. 5(a). Our aim is to compute the outgoing messages from check node k back to dcadjacent bit nodes. Let us denote these messages by $w_{k \to n_1}$, $w_{k \to n_2}$,$\cdots$, $w_{k \to n_{dc}}$. Each outgoing message from check node k to its adjacent bit nodes is computed as

$$w_{k\to n_i} = g\left(v_{n_1\to k}, v_{n_2\to k},\ldots, v_{n_{i-1}\to k}, v_{n_{i+1}\to k},\ldots, v_{n_{dc}\to k}\right),$$
$$g(a,b) = \mathrm{sgn}(a)\times\mathrm{sgn}(b)\times\min(|a|,|b|) + LUT_g(a,b),$$
$$LUT_g(a,b) = \log(1+e^{-|a+b|}) - \log(1+e^{-|a-b|}). \quad (5)$$

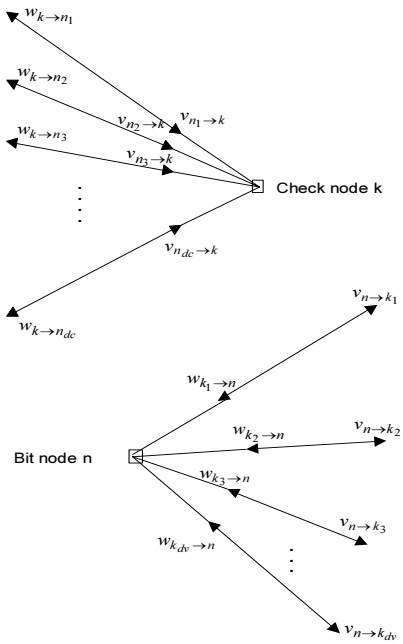


Fig. 5. Message update at (a) check nodes and (b) bit nodes

In practice, the $LUT_g(.)$ function is implemented using a small look-up table.

Step 3. Bit Node Update

Let us denote the incoming messages to bit node n from its dv adjacent check nodes by $w_{k_1\to n}$, $w_{k_2\to n}$,..., $w_{k_{dv}\to n}$, as shown in Fig. 5(b). Our aim is to compute the outgoing messages from bit node n back to dv adjacent check nodes. Let us denote these messages by $v_{n\to k_1}$, $v_{n\to k_2}$,..., $v_{n\to k_{dv}}$. They are computed as

$$v_{n\to k_i} = u_n + \sum_{j\neq i} w_{k_j\to n}. \quad (6)$$

Intuitively, this is a soft majority vote on the value of bit n, using all relevant information except $w_{k_j\to n}$

3.2 Sequential Decoding Algorithm with Partial Group

In this paper, we propose a new decoding structure. First, we divide check nodes by p groups. The groups are in general randomly chosen so as to minimize the cycle-4 occurrence. Next, we decode group by group in a serial fashion. For example, Fig. 6 shows the decoding procedure in the case when p equals 2. Check node and bit node probabilities are calculated at the first group, as shown in Fig. 6(a). When the decoding process of the first group has been completed, the second group initiates the decoding process with the calculated bit node probability from the first group, as shown in Fig. 6(b). The proposed algorithm changes only the decoding order without any performance degradation.

If p equals one, the result is the same as a conventional decoder. Like the turbo decoding, the resultant extrinsic information of the first decoder is delivered to the second decoder. The proposed algorithm of LDPC decoding is the same theory.

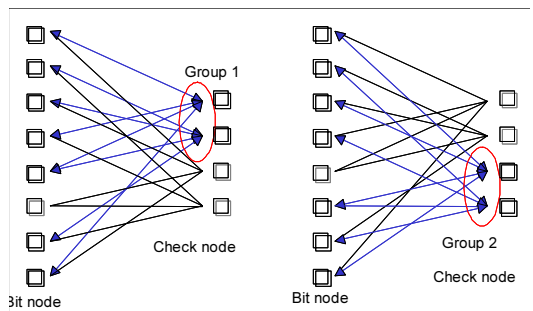


Fig. 6. The bit and check node update procedure (p=2) at (a) the first group (b) the second group

3.3 Early Edge Detection Algorithm

Another approach is early edge detection for reducing the computational complexity. The early edge detection method is based on the observation that bit nodes and check nodes with a high log-likelihood value can be considered reliable. So detected bit node edges and check node edges are negligible for the next iteration. This means that

we don't need to calculate the bit node edges and check node edge updates for a detected edge from following iterations. Therefore, computational complexity of the next iterations is reduced. For early edge detection for the LDPC decoder, the proceedings are as follows:

Step 1. Initialization

$\text{EarlyDetectCheck}[k \to n_i] = 0$

$\text{EarlyDetectBit}[n \to k_i] = 0$

Step 2. Check node updates

if $w_{k \to n_i} \geq T_c$, then $\text{EarlyDetectCheck}[k \to n_i] = 1$

else do *Eq.* (5)

Step 3. Bit node updates

$v_{n \to k_i} \geq T_b$, then $\text{EarlyDetectBit}[n \to k_i] = 1$

else do *Eq.* (6)

If *EarlyDetectCheck*[·] or *EarlyDetectBit*[·] are equal to one, we don't need to calculate the bit node and check node updates, we just deliver the previous values to the bit nodes or check nodes. Therefore, it is very important to choose the early detection threshold: T_b for bit nodes and T_c for check nodes.

3.4 Simulation Results

Fig. 7 shows the simulation results with codeword size N=64800 and information size K=32400 (number of check node, M=N_K), comparing the BER performances of the conventional and sequential LDPC decoders. With a smaller number of iterations (N_r), the sequential decoding scheme with p = 2 (N_r = 35) shows the performance almost the same as that of a conventional scheme with p = 1 (N_r = 70). Fixing on T_c=10, we evaluated the performance for various values of T_b. As shown in Fig. 8, the performance of the early detection method with T_b=18 is the same as that of a conventional scheme.

The required number of operations associated with the conventional and early detection algorithms are summarized in Table 2, where d_c^* denotes the average number of early detected edges for each check node and N^* denotes the average number of early detected bit nodes. The decoding complexity of an LDPC decoder is evaluated based on Eq.(5) and Eq.(6).

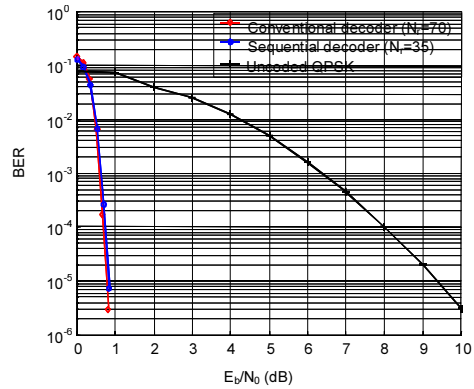


Fig. 7. Bit error rate (BER) comparison between conventional LDPC decoder and sequential LDPC decoder(p=2)

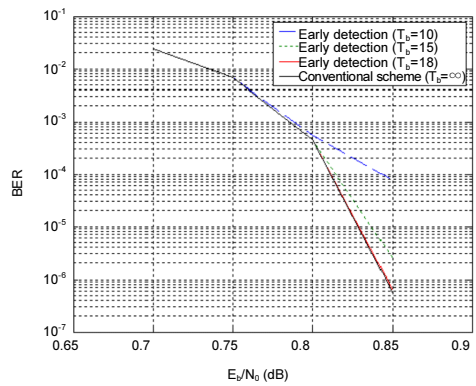


Fig. 8. BER performance for different T_b (N=64800, K=32400 d_c=7, d_v=13

Fixing on the values of T_b =18 and T_c =10, Table 3 shows the simulated d_c^* and N^* from N_r iterations at E_b/N_0=1 (dB) in the environment of N=64800, K=32400, M=32400, and d_c=7, d_v=13. As shown in Table 2, the computational complexity of the early detected method is about 50% off in the case of the check node update, and 99% off in the case of the check node update compared to the standard LDPC decoder scheme.

Table 2. Decoding complexity of conventioanl and early detection methods

	Conventional decoder	Early detection method
Check nodes ($g(a,b)$)	$(M \times (d_c - 1)) \times N_r$	$(M \times (d_c - d_c^* - 1)) \times N_r$
Bit nodes (additions)	$(N \times (d_v - 1)) \times N_r$	$((N - N^*) \times d_v) \times N_r$

Table 3. The number of early detected edges

	N_r					
	10	20	30	40	50	60
d_c^*	0.387	1.484	2.619	**3.29**	**3.6**	**3.8**
N^*	491	19048	62372	**64334**	**64335**	**64335**

Ⅳ. Ⅰ. Low Latency Algorithms of TPC codes

The real difficulty in the field of channel coding is essentially a problem of decoding complexity of powerful codes. Recently, there has been intensive focus on turbo product code (TPC) which has low latency and simple structures compare with turbo code. It can achieve performances near Shannon limit. TPCs are two dimensional code constructed from small component codes. A two dimensional turbo product code(TPC) can be noted as C1⊗C2, where C1 and C2 are two linear block codes. Place k1⊗k2 information symbols in array of k1 rows and k2 columns, and then encode the k1 rows using code C2. Afterwards, the resulting k2 columns are encoded using code C1. Usually, we choose C1 the same as C2. The conventional TPC decoder performs row and column decoding in a serial fashion. A soft input soft output(SISO) decoder, such as MAP, is used to decode each row or column. The output of decoder is the reliability of the decision d_j and the relationship expression between soft-out r_j' and soft-in r_j is given by $r_j'=r_j+\alpha w_j$. The extrinsic information w_j for the j-th bit position is given by $w_j=\beta d_j$. Where β is a reliability factor to estimate w_j in case no competing codeword can be found. a is a weight factor to combat high standard deviation in w_j and high BER during the first iterations. Fig.9 shows encoder construction and decoder structure.

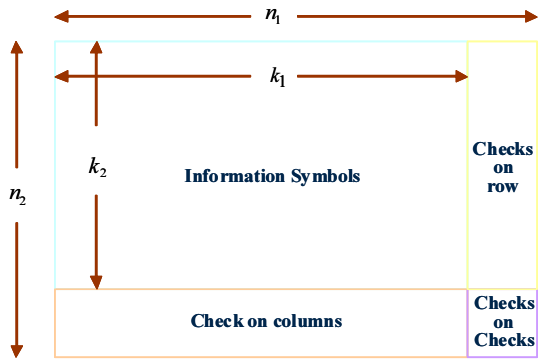
The operations above are performed on all bits of a product codeword, shown in Fig 2., hence, Eq.(1) can be expressed in matrix form as

$$R[2]=[R]+\alpha[2]\cdot[W[2]] \quad . \qquad (7)$$

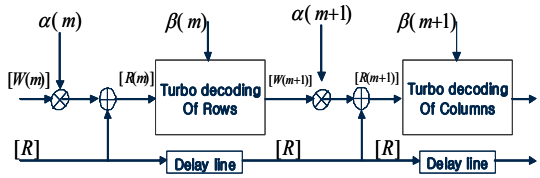
For decoding at step m, we used the following values

$\alpha(m)=[0,\ 0.2,\ 0.3,\ 0.5,\ 0.7,\ 0.9,\ 1.0,\ 1.0]$,

$\beta(m)=[0.2,\ 0.4,\ 0.6,\ 0.8,\ 1.0,\ 1.0,\ 1.0,\ 1.0]$.



(a) Encoder construction



(b) Conventional serial TPC decoder

Fig. 9. Encoder construction and decoder structure of turbo product code ($P=C_1\times C_2$)

4.1 P-Parallel Algorithm

A low complexity decoding approach is provided in [13]. It applies the Chase algorithm iteratively on the row and column decoding [7], but still in a serial fashion. In order to halve the decoding latency, a p-parallel TPC decoder is proposed in this paper. As opposed to the conventional serial TPC decoder, the row and column decoders operate in parallel and update each other immediately after a row and column has been decoded at the same time. Different than in the conventional serial TPC decoder, decoding time of the proposed algorithm is halved. Furthermore, the whole product codeword needs to be decoded row-wise or column-wise for N times before next iteration can begin, where N equals the column number and the row number of product code array. P-Parallel TPC decoder is a parallel decoding scheme combining that p-rows and p-columns of Chase decoder are processed in parallel instead of decoding one by one as that in the original

scheme. For the decoder shown in Fig.9(b), the conventional TPC decoder needs to decode row or column before the next half-iteration can begin. Row and column decoders operate in parallel and update each other immediately after a row and column has been decoded at the same time to reduce the latency to halve the decoding time as shown in Fig.10.

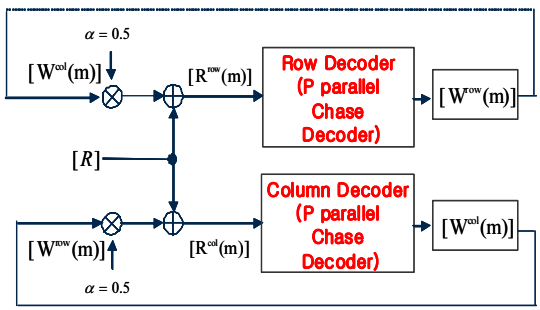


Fig.10 P-parallel decoding structure

The matrices $[W^{row}]$ and $[W^{col}]$, which are the row and column extrinsic information matrices, are passed to row or column decoder by block basis simultaneously. For the first iteration at m=0, we set $[R^{row}(0)]=[R^{col}(0)]=[R]$. Let us assume the same code of block length n is used as the row and column code of the product code. As soon as the row and column decoder have finished decoding of all rows and columns, respectively, they pass their updated matrices $[R^{row}(m+1)]$and $[R^{col}(m+1)]$ as inputs to the next decoding stage. As a result of parallel updating, update metric may be written as

$$[R^{row}(m+1)]=[R]+\alpha(m)\cdot[W^{col}(m)]$$
$$[R^{col}(m+1)]=[R]+\alpha(m)\cdot[W^{row}(m)]. \quad (8)$$

The weight factor and reliability factor for each iteration m, we search optimal values by computer simulation.

$$\beta(m)=\{0.5,0.5,0.5,0.5,1.0,1.0,1.0,1.0,1.0,1.0\}$$
$$\alpha(m)=0.5 \text{ for } \forall m.$$

4.2 Early Stop Algorithm

HAD algorithm is also used for early-stopping algorithm. Compare each decision that was generated during the row/column Chase decoder. When the two sets of decisions match, stop decoding on the current block and return the hard decision bits as output.

Table 4. The average of iterations according to E_b/N_0 (the predetermined number of iterations id 6)

E_b/N_o	Average number of iterations	Decoding speed improvement (%)
1	6	0
2	5.986	0.2
3	4.82	19.7
4	2.049	65.85

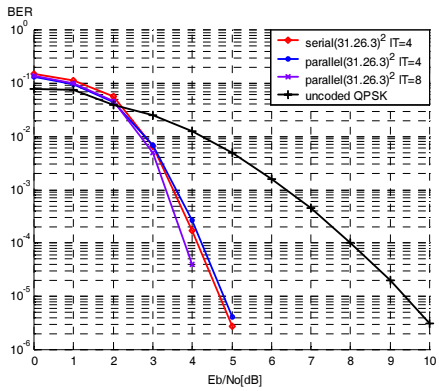
Table IV shows average iteration number in applying HDA algorithm to parallel TPC decoder with coding rate of $(31,26,3)^2$and QPSK modulation scheme. At EbNo is 4[dB], it just required about 2 iterations maintain same performance level. It means that decoding speed is improved about 66% or power consumption (cost) is 66% off.

4.3 Simulation Results

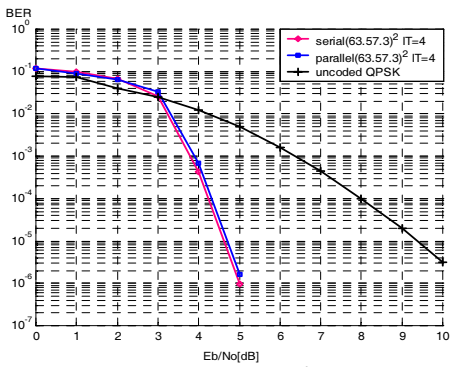
The results that we present here concern BCH product code with several code rates. Especially, TPCs with one-error correcting BCH component codes are suitable for applications which require both high data and code rates. Serial (n,k,δ) is TPC decoder which is used to serially two BCH decoder in row and column decoder. P-parallel (n,k,δ) is TPC decoder which is used to parallel two BCH decoder at row and column decoder. For chase decoding, the number of least reliable bits was chosen to be 4. Our simulation results for two TPCs $((31.26.3)^2,(63.57.3)^2)$, with four decoding iterations are given in Fig.11.

We observe that the performance of p-parallel TPC decoder(p=2) is similar to that of conventional TPC decoder at BER=10^{-4}. In Fig.11, we also included the performance of a parallel (31.26.3) with eight iterations. This TPC has the same decoding latency as the conventional decod-

ing approach with four iterations, but achieves an additional gain of 0.4dB at BER=10^{-4}.



(a) TPC $(31.26.3)^2$



(b) TPC $(63.57.3)^2$

Fig. 11 Performance of serial and parallel decoded BCH TPCs (p=2, IT denotes number of iteration)

Ⅴ. Conclusions

Low latency versions of iterative decoders of turbo codes, low-density parity-check codes and turbo product codes are presented. Among of the latency algorithms, early-stop and parallel fashioned algorithms may be applied to all of the iterative codes in common.

Radix-4 and dual-path processing algorithms for turbo decoding, sequential method and early edge detection algorithms for LDPC decoding, and p-parallel structure for TPC decoding are applied in order to reduce the latency and/or computational complexity. From the performance analysis by computer simulation and average number of eliminated edge or iterations, we conclude that the presented algorithms provides an attractive solution to implementation iterative decoding in aspect to fast decoding speed and power consumption.

References

〔1〕 C. Berrou, A. Glavieux, and P.Thitimajshima, "Near Shannon Limit Error-Correcting Code and Decoding : Turbo Codes", *IEEE Trans. Commun.*, vol. 44, pp.1261-1271,1998.

〔2〕 D. J. C. Mackay and R. M. Neal, "Near Shannon Limit Performance of Low-Density Parity-Check Codes,"*Electron. Letter*, Vol.32, PP. 1645-1646,Aug.1996.

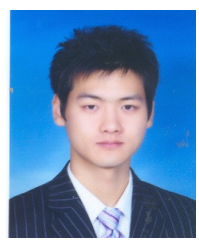
〔3〕 R.M. Pyndiah, "Near-optimum decoding of product codes: Block turbo codes," *IEEE Trans. Commun.*, vol. 46, pp1003-1010, Aug. 1998.

〔4〕 S.S. Pietroloon, "Implementation and Performance of a Turbo/MAP Decoder", *International Journal of Satellite Communication* vol. 16, pp.23-46, 1998.

〔5〕 J.Zhang and M.Fossorier, " Shuffled Belief Propagation Decoding," *IEEE Trans. Commun.*, Feb. 2005.

〔6〕 D.Chase, "A class of algorithms for decoding block codes with channel measurement information," *IEEE Trans.Inform. Theory*, vol. IT-18,pp.170-182,Jan.1972

최 석 순 (Seok Soon Choi) 준회원



2007년 2월: 한국해양대학교 전파공학과(공학사)

2007년 3월 ~ 현재: 한국해양대학교 전파공학과 석사과정

<관심분야> 위성통신, 이동통신, 변·복조기술, 채널코딩, FPGA 기술 등

정 지 원 (Ji Won Jung) 정회원

1989년 2월 :성균관대학교 전자공학과(공학사)
1991년 2월 :성균관대학교 전자공학과(공학석사)
1995년 2월 :성균관대학교 정보공학과(공학박사)
1991년 1월～ 1992년 2월 : LG 정보통신연구소 연구원
1995년 9월 ～ 1996년 8월 : 한국통신 위성통신연구실 선임연구원
1997년 3월 ～ 1998년12월 : 한국전자통신연구원 초빙연구원
1996년 9월 ～ 현재: 한국해양대학교 전파공학과 정교수
2001년 8월 ～ 2002년 8월 : 캐나다 NSERC Fellowship (Communication Research Center 근무)
<관심분야> 위성통신, 이동통신, 변.복조기술, 채널코딩, FPGA 기술 등

배 종 태 (Jong Tae Bae) 준회원

2007년 2월: 한국해양대학교 전파공학과(공학사)
2007년 3월 ~ 현재: 한국해양대학교 전파공학과 석사과정
<관심분야> 위성통신, 이동통신, 변·복조기술, 채널코딩, FPGA 기술 등

김 민 혁 (Min Hyuk Kim) 준회원

2006년 2월: 한국해양대학교 전파공학과(공학사)
2006년 3월 ~ 현재: 한국해양대학교 전파공학과 석사과정
<관심분야> 위성통신, 이동통신, 변·복조기술, 채널코딩, FPGA 기술 등

최 은 아 (Eun A Choi) 정회원

1998년 2월: 전북대학교 수학과(이학사)
2000년 2월: 전북대학교 대학원 정보통신공학과(공학석사)
2000년 4월 ~ 현재: 한국전자통신연구원 광대역 멀티미디어 연구팀 선임 연구원
<관심분야> 채널코딩, 디지털통신, 위성통신 등