Scheduling Algorithm for Partially Parallel Architecture of LDPC Decoder by Matrix Permutation

In-Cheol Park and Se-Hyeon Kang
Department of Electrical Engineering and Computer Science, KAIST
373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, Republic of Korea
{icpark, shkang}@ics.kaist.ac.kr

Abstract— **The fully parallel LDPC decoding architecture can achieve high decoding throughput, but it suffers from large hardware complexity caused by a large set of processing units and complex interconnections. A practical solution of area-efficient decoders is to use the partially parallel architecture in which a PU is shared for a several rows or columns. It is important in the partially parallel architecture to determine the rows or columns to be processed in a PU and their processing order. The dependencies between rows and columns should be considered to minimize the overall processing time by overlapping the decoding operations. This paper proposes an efficient scheduling algorithm that can be applied to general LDPC codes, which is based on the concept of the matrix permutation. Experimental results show that the proposed scheduling achieves a higher decoding rate, leading to a reduction of 25% processing time on the average. A 1024-bit rate-1/2 LDPC decoder employing the proposed scheduling algorithm provides almost 1Gbps decoding throughput and occupies one-fifth area compared to the fully parallel decoder.**

I. INTRODUCTION

Low density parity check (LDPC) codes are originally devised to exploit low decoding complexity by constructing sparse parity check matrices. Though the LDPC code does not have a maximized minimum distance due to the randomly generated sparse parity check matrix, the typical minimum distance increases linearly as the block length increases. Moreover, the error probability decreases exponentially for a sufficiently long block length, whereas the decoding complexity is linearly proportional to the code length. Recent simulation results show that the LDPC code can achieve a performance that is within 0.04 dB of Shannon limit and the performance is close to that of the turbo code if the block length is larger than 1000 bits [2].

Despite of these advantages, when the LDPC code was first introduced, it made a little impact on the information theory community because of the storage requirements in encoding and the computational complexity in decoding. Modern VLSI technology is so advanced that it enables parallel architectures exploiting the benefit of inherently parallel LDPC decoding algorithms. Blanksby et al. implemented an 1-Gb/s fully parallel decoder in which the message passing algorithm is directly mapped [3]. This architecture, however, requires a large number of complex routings between concurrent processing units (PUs) each of which corresponds to a node of the message passing algorithm, leading to the average net length of 3mm and the total die size of 52.5mm^2. On the other side, Yeo et al. proposed an area-efficient architecture that serializes the computations by sharing PUs [4]. Consequently, one iteration takes 9612 cycles and wide-input multiplexers are required to select one of 18432 intermediate values to be fed into the shared PUs. These two counter examples show that high throughput LDPC decoding architectures should exploit the benefit of parallel decoding algorithms while reducing the interconnection complexity.

The partially parallel architecture is a good trade-off between throughput and hardware cost. Since a PU is shared for a number of rows or columns, the number of PUs becomes much smaller than that of the fully parallel architecture. As decoding operations are parallel in nature, it is important to determine which rows or columns are processed in a PU. In the grouping, the dependencies between rows and columns should be considered to minimize the overall cycles by overlapping the decoding operations. There has been a heuristic scheduling algorithm proposed for quasi-cyclic LDPC codes [9], but it cannot be applied to general LDPC codes. This paper proposes an efficient scheduling algorithm that can be applied to general LDPC codes. The proposed algorithm is based on the concept of the matrix permutation.

The rest of this paper is organized as follows. Section II explains briefly about the low density parity check code and the decoding algorithm. Section III proposes a new scheduling algorithm which helps partially parallel LDPC decoder to achieve high throughput using matrix permutation. Experimental results of the proposed algorithm are shown in Section IV. Finally, Section V addresses some concluding remarks.

II. LDPC DECODER ARCHITECTURE

The LDPC code, which was first introduced by Gallager in 1962 [1], is defined by a binary linear block code of length n and a parity check matrix H with a column weight γ and a row weight ρ: (n, γ, ρ). The parity check matrix H has n columns and J rows that correspond to the block length and the total number of parity check equations of the code, respectively. The column weight γ and the row weight ρ represent the number of 1's in a column and a row, respectively, and they are much smaller than n to achieve the sparse matrix H. Fig. 1 (a) shows an example matrix H of (12, 3, 6) LDPC code with indicating the code parameters.

A. Message Passing Algorithm

The LDPC code is often represented by a factor graph to make it easy to understand the message passing decoding algorithm, which is a bipartite graph that expresses how a global function of

0-7803-8834-8/05/$20.00 ©2005 IEEE.

many variables is factored into a product of local functions [7]. Fig. 1 (b) shows the factor graph of a (12, 3, 6) LDPC code, which consists of two sets of nodes: i.e. variable nodes, $\{v_j\}$, and check nodes, $\{c_i\}$. The edge between a variable node v_j and a check node c_i is constructed if there is 1 at (i, j) in the parity check matrix. Therefore each check node represents a check equation used in generating parity check bits and each variable node represents one bit in the codeword. The variable nodes that are connected to a check node are called the neighbor variable nodes of the check node. For a variable node, the neighbor check nodes are defined similarly.

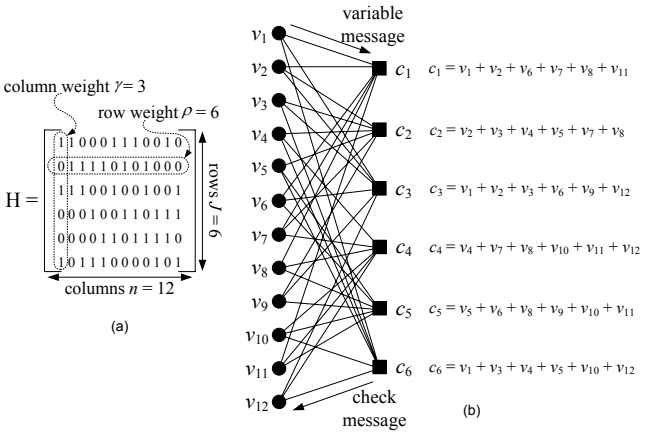

Fig. 1. A (12, 3, 6) LDPC code. (a) Parity check matrix. (b) Factor graph.

Unlike the general parity check codes, a LDPC code cannot be optimally decoded since its decoding is NP-complete. An iterative approximate algorithm called a message-passing algorithm is used instead, which is also known as sum-product [8] or belief propagation [5]. The message-passing algorithm can be expressed easily based on the factor graph as below.

1) Initialize each variable node v_j to the probability ratio of the corresponding received bit. The probability ratio is the first variable message to be sent to the neighbor check nodes.

$$\Delta_{ij} = \Delta_j = \frac{p(r_j \mid x_j = +1)}{P(r_j \mid x_j = -1)} = \exp(\frac{2r_j}{\sigma^2}) \tag{1}$$

2) Variable-to-check (VTC) step: Each check node c_i computes the likelihood ratio, Λ_{ij}, by using variable messages of its neighbor variable nodes except from v_j. This is the check message to be sent to variable node v_j.

$$\Lambda_{ij} = \prod_{j' \in N(i) \setminus j} \frac{1-\Delta_{ij'}}{1+\Delta_{ij'}} \tag{2}$$

3) Check-to-variable (CTV) step: Each variable node v_j computes probability ratios, denoted by Δ_{ij}, by using check messages of its neighbor check nodes except from c_i. This is the variable message to be sent again to check node c_i.

$$\Delta_{ij} = \frac{p(r_j \mid +1)}{P(r_j \mid -1)} \prod_{i' \in M(j) \setminus i} \frac{1-\Lambda_{i'j}}{1+\Lambda_{i'j}} \tag{3}$$

4) For each variable node, create a tentative bit-by-bit decoding x_j using the pseudo-posteriori probability Δ_j.

$$\Delta_j = \frac{p(r_j \mid +1)}{P(r_j \mid -1)} \prod_{i \in M(j)} \frac{1-\Lambda_{ij}}{1+\Lambda_{ij}} \tag{4}$$

$$x_j = \begin{cases} 0 & when\ \Delta_j \le 1 \\ 1 & when\ \Delta_j > 1 \end{cases} \tag{5}$$

5) Check if $x \cdot H^T = 0$ is satisfied. If it is satisfied or the maximum number of iterations is reached, the decoding algorithm finishes. Otherwise, the algorithm repeats from step 2).

The symbols, Δ and Λ, correspond to the outgoing messages of the variable and check nodes, respectively, and $N(i)$ and $M(j)$ represent the set of neighbor nodes of check node c_i and variable node v_j, respectively. The not-notation (') marked as superscript means that the product is calculated over the indexes excluding its own index. The VTC operation calculates the check messages $\{\Lambda_{ij}\}$ of check node c_i using the variable messages $\{\Delta_{ij}\}$ for variable nodes identified by the i-th row of the matrix H. A detailed explanation of the algorithm can be found in [6].

B. Partially Parallel Architecture

The fully parallel architecture is inherited from the message-passing algorithm and all the operations required in each node are implemented as a PU. The number of PUs is as many as the number of the variable and check nodes. Although the fully parallel implementation of the message-passing algorithm is straightforward and results in a high throughput decoder, it faces with complex interconnections required to sum up the Λ_{ij} and Δ_{ij} messages that are calculated in the PUs spread over the chip area.

In the partially parallel architecture, each PU takes in charge of several numbers of rows or columns as shown in Fig. 2. As a PU is shared for a number of rows or columns, the number of PUs becomes much smaller than that of the fully parallel architecture. Thus the number of VTC (CTV) operations processed in a cycle is as many as the number of check (variable) PUs. The check messages calculated row-by-row by a check PU can be grouped and stored into a local memory to save area, and variable PUs access them later.

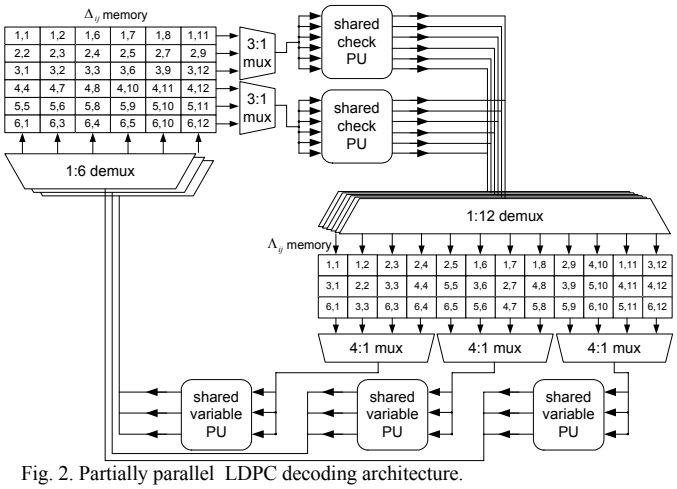

Fig. 2. Partially parallel LDPC decoding architecture.

The VTC and CTV operations are parallel in nature and thus a PU can process any rows or columns regardless of their order in matrix H. Since a PU takes in charge of several rows or columns in the partially parallel architecture, we have to determine which rows or columns are processed in the PU. In the grouping, the dependencies between rows and columns should be considered to minimize overall cycles by overlapping the VTC and CTV operations. For example, the CTV operation for v_1 in the (12, 3, 6) LDPC code requires the VTC operations for c_1, c_3 and c_6 to be completed beforehand. In addition, it is important to determine the processing order of rows or columns for a PU, because the dependence of the Λ_{ij} and Δ_{ij} can hinder the overlapped processing.

III. Scheduling Algorithm

Traditionally, the CTV operations can start only after the entire VTC step finishes completely, and vice versa. A well-scheduled

sequence of the message calculations can reduce the latency, because a CTV operation can start in the course of the VTC step if the corresponding row summation values are available, and vice versa. The overlapped processing of the VTC and CTV steps results in a reduced number of cycles for an iteration.

A. Overlapped Processing

Fig. 3 shows an example of overlapped processing for the (12, 3, 6) LDPC code, assuming that the numbers of check PUs and variable PUs are two and three, respectively. Each arrow denotes a clock cycle and the numbers above an arrow indicate the row or column indices processed at that cycle. If the VTC and CTV steps are performed according to the increasing order of indices without scheduling, no overlapped processing is possible. Though two CTV operations for v_2 and v_7 can start at the last cycle of the VTC step, the CTV step does not start until three CTV operations are enabled for easy control design. If the VTC operations are scheduled as shown in Fig. 4 (b), three CTV operations for v_2, v_6 and v_9 can start processing at the last cycle of the VTC step. Furthermore, the CTV operations can be scheduled to enable the VTC operations of the next iteration to start earlier. In this example, the scheduling of the VTC and CTV operations saves two cycles.

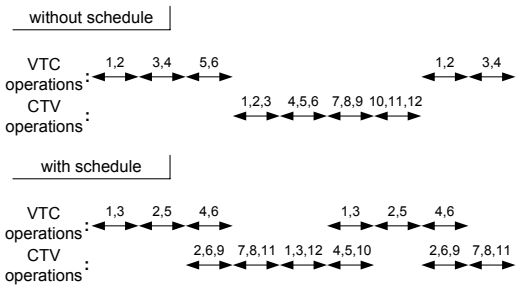

Fig. 3. An example of overlapped processing

The scheduling process can be viewed as the row and column permutation of the parity check matrix H. Fig. 4 (a) is the original parity check matrix H whose size is $M \times N$. In this case, the size of H is 6×12 and we assume 2 check PUs and 3 variable PUs. If there are m check PUs, m rows at the top of the matrix are processed in the first cycle, the next m rows in the second cycle, and so on. Thus the VTC step takes M/m cycles. Similarly, the CTV step takes N/n cycles if there are n variable PUs. The permutation changes the sequence of the VTC and CTV operations to enlarge the empty slots at the lower left and upper right corners as shaded in Fig. 4 (b).

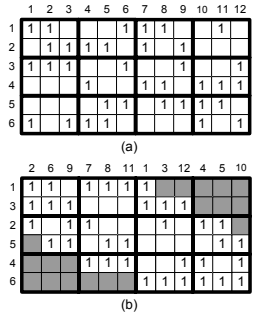

Fig. 4. Scheduling by the permutation of matrix H. (a) Matrix H. (b) Permuted matrix.

The empty slots in a column mean that the processing of the column can start as long as the rows above the empty slots are processed beforehand. If there are k empty slots in a column, the column processing can start $\lfloor k/m \rfloor$ cycles earlier, and if the first n columns have more than m empty slots, all the variable PUs can start their processing one cycle earlier. The simultaneous early starting is possible if the $m \times n$ slots that are at the lower left part surrounded by thick lines are all empty, and it is required to achieve an easy control design. For example, as three columns have empty slots larger than two in Fig. 4 (b), they can start after two cycles of the VTC step. Similarly, the empty slots in a row mean that the processing of the row can start as long as the columns at the left hand side of the empty slots are processed beforehand, leading to two rows that can start after three cycles of the CTV step.

In fact, the earliest cycle found by the permutation is not the real starting time of the CTV (VTC) step. For the sake of easy control design, we assume that the CTV (VTC) step performs continuously without skipping cycles once it starts. If there is a cycle in which any column (row) cannot be processed during the CTV (VTC) step, the former operations are delayed to achieve continuous CTV (VTC) operations.

B. Scheduling Algorithm

It takes a considerable amount of time to find an optimum schedule that minimizes the overall cycles because of the large number of rows and columns. There has been a heuristic scheduling algorithm proposed for quasi-cyclic LDPC codes [9], but it cannot be applied to general LDPC codes. We describe a new scheduling algorithm developed for the partially parallel LDPC decoding architecture. The proposed algorithm described below is based on the concept of the matrix permutation. The algorithm makes the row sequence and column sequence that can result in empty spaces in the lower left and upper right corners of the permuted matrix when the matrix H is rearranged according to the sequences.

Step 1: row sequence = null, column sequence = null.

Step 2: Select a row r_k randomly.
The row sequence is enlarged by appending r_k, and the column sequence is appended by inserting $\{c_y | c_y \in r_k\}$. Go to Step 4.

Step 3: If all the rows contained in a column are selected, such a column is called a completed column. If a column contained in a row is also in the column sequence, it is called a common column of the row. For each unselected row, count the number of common columns (CC) and the number of columns to be completed (CTC) when the row is selected. Select a row from unselected rows, r_i, that has the maximal CTC. If there are more than two rows associated with the maximal CTC, we select the one that has the largest CC.
The row sequence is appended by r_i, and the column sequence is enlarged by appending the columns of r_i that are not in the current column sequence.

Step 4: Find completed columns, and move them to the front of the first uncompleted column in the column sequence.

Step 5: If there are unselected rows, go to Step 3.

Step 6: Permute the matrix H according to the row sequence and column sequence.

Step 7: If there are m check PUs, the first m rows at the top of the permuted matrix are assigned to the first cycle of the VTC step, the next m rows to the second cycle, and so on. Similarly, for n variable PUs, the first n columns at the left of the permuted matrix are assigned to the first cycle of the CTV step, the next n columns to the second cycle, and so on.

The random row selection in Step 2 is to explore more search space by starting from a different row. Step 3 to 5 are the main loop for matrix permutation and Step 6 to 7 assign rows and columns to PUs considering easy control design after scheduling. In the algorithm, rows are selected to make the upper right part of the permuted matrix as empty as possible and completed columns are moved to the left in order to make enlarged empty space in the lower left part. An example of the algorithm is illustrated in Fig. 5.

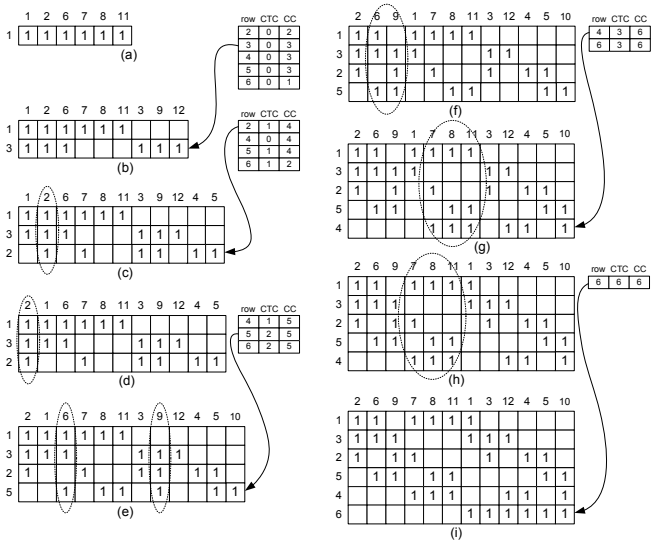

Fig. 5. An illustrative example of the proposed scheduling. (a) Select r_1. (b) Append r_3. (c) Append r_2. (d) Move c_2. (e) Append r_5. (f) Move c_6, c_9. (g) Append r_4. (h) Move c_7, c_8, c_{11}. (i) Append r_6.

IV. Experimental Results

By applying the scheduling algorithm to a (1024, 3, 6) LDPC code, we can save the number of processing cycles per iteration as summarized in Table I, where parallelism (m, n) stands for an implementation associated with m check PUs and n variable PUs.

TABLE I
CYCLES SAVED BY SCHEDULING PER ITERATION

Parallelism	Cycles w/o schedule	Cycles w/ schedule	Saved cycles	Saved cycles (%)
(4, 8)	256	195	61	23.8
(8, 8)	192	139	53	27.6
(16, 8)	160	129	31	19.4
(8, 16)	128	93	35	27.3
(16, 16)	96	70	26	27.1
(32, 16)	80	60	20	25.0
(16, 32)	64	47	17	26.6
(32, 32)	48	36	12	25.0
(64, 32)	40	30	10	25.0

We designed two (1024, 3, 6) LDPC decoders using a 0.18 um CMOS process for parallelism (16, 32) and (32, 64). The performances of the decoders are summarized in Table II. The first decoder corresponding to parallelism (16, 32) occupies an area of 6.29 mm2 and provides more than 500 Mbps decoding throughput at the frequency of 200 MHz, and the second decoder corresponding to parallelism (32, 64) takes 10.08 mm2 and provides almost 1Gbps decoding throughput at the same frequency. The resulting bit rate of the second decoder is comparable to the fully parallel architecture proposed by Blanksby et al., but significant area reduction is achieved. Although the iteration number in Blanksby's implementation is fixed to 64 due to the scan-chain like I/O mechanism, there is no restriction in the proposed architecture. We chose 8 iterations to provide sufficient BER performance.

TABLE II
COMPARISON OF LDPC DECODERS

	Blanksby [3]	Proposed	
		Parallelism (16,32)	Parallelism (32,64)
Technology	0.16 um	0.18 um	0.18 um
Bit rate	1 Gbps	582 Mbps	985 Mbps
Area	52.5 mm^2	6.29 mm^2	10.08 mm^2
		MEM: 3.46 mm^2	6.45 mm^2
		Other: 2.83 mm^2	3.63 mm^2

V. Conclusions

This paper has presented an efficient scheduling algorithm for partially parallel LDPC decoders to minimize the overall processing time by overlapping CTV and VTC steps. As the VTC and CTV operations are parallel in nature, a PU can process any rows or columns regardless of their order in matrix H, and thus it is very important to determine which rows or columns are processed in a PU. In the grouping, the dependencies between rows and columns should be considered to overlap the VTC and CTV operations. Based on the concept of the matrix permutation, the proposed scheduling algorithm generates the row sequence and column sequence that can result in empty spaces in the lower left and upper right corners of the permuted matrix when matrix H is rearranged according to the sequences. Experimental results show that the proposed scheduling achieves a higher decoding rate, leading to a reduction of 25% processing time on the average.

Acknowledgment

This work was supported by Institute of Information Technology Assessment through the ITRC, by the Korea Science and Engineering Foundation through MICROS center and by IC Design Education Center (IDEC).

References

[1] R. G. Gallager, "Low density parity check codes," *IRE Trans. Info. Theory*, vol. IT-8, pp. 533-547, Jan. 1962.

[2] S. Chung, D. Forney, T. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 db of the Shannon limit," *IEEE Comm. Letters*, vol. 5, pp. 58-60, Feb. 2001.

[3] A. Blanksby and C. Howland, "A 690-mW 1-Gb/s, rate-1/2 low-density parity-check code decoder," *IEEE J. of Solid-State Circuits*, vol. 37, pp. 404-412, Mar. 2002.

[4] E. Yeo, P. Pakzad, B. Nikolić and V. Anantharam, "VLSI architectures for iterative decoders in magnetic recording channels," *IEEE Trans. Magnetics*, vol. 37, pp. 748-755, Mar. 2001.

[5] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1988.

[6] K. Lo, *Layered space time structures with low density parity check and convolutional codes*, MS Thesis, School of EIE, Univ. of Sydney, 2001.

[7] F. R. Kschischang, B. J. Frey, and H. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Info. Theory*, vol. 47, pp. 498-519, Feb. 2001.

[8] N. Wiberg, *Codes and decoding on general graphs*, PhD thesis, Dept. of EE, Linköping Univ. Sweden, 1996.

[9] Y. Chen, K. K. Parhi, "Overlapped message passing for quasi-cyclic low-density parity check codes," *IEEE Trans. Circuits and Syst. I*, vol. 51, pp. 1106-1113, Jun. 2004.