Memory-Based Low Density Parity Check Code Decoder Architecture Using Loosely Coupled Two Data Flows

Se-Hyeon and In-Cheol Park
Department of Electrical Engineering and Computer Science, KAIST
373-1 Gusung-Dong Yuseong-Gu, Daejeon 305-701, Republic of Korea

Email: shkang@ics.kaist.ac.kr,icpark@ee.kaist.ac.kr

Abstract — *To achieve high throughput, parallel decoding of low density parity check (LDPC) codes is required, but needs a large set of registers and complex interconnection due to randomly located 1's in a sparse parity check matrix of large block size. This paper proposes a memory-based decoding architecture for low density parity check codes using loosely coupled two data flows. Instead of register, intermediate values are optimally grouped and scheduled to store into the segmented memory, which reduces large area and enables a scalable architecture. The performance of the proposed decoder architecture is demonstrated by implementing a 1024 bit, rate-1/2 LDPC codes decoder.*

I. Introduction

Low density parity check (LDPC) codes are originally devised to exploit low decoding complexity by constructing a sparse parity check matrix. Since LDPC codewords does not have a maximized minimum distance due to the sparse parity check matrix, the typical minimum distance increases linearly with the block length. The error probability decreases exponentially for sufficiently long block length. Moreover the decoding complexity is linearly proportional to the code length. LDPC codes are applied to various areas such as data storage, digital subscriber line and CDMA, especially space time coded OFDM systems due to its scalability [8] [9] [10] [11]. Recent simulation results show that the performance of LDPC codes is close to that of turbo codes [5] [6].

Despite of these advantages, LDPC codes made little impact on the information theory community because of the storage requirements in encoding and the computational demands of decoding. The Modern VLSI technology enables decoding architectures to exploit the benefit of inherently parallel decoding algorithm for LDPC codes. Blanksby et al. implemented a 1-Gb/s fully parallel decoder in which the message passing algorithm is directly mapped [7]. This architecture, however, requires complex hand-wired routing between concurrent processing elements corresponding to each 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 which serializes the computations of each row and column by sharing computation units. Consequently, one iteration in decoding a codeword takes 9612 cycles and wide input multiplexers are required to select one of 18432 intermediate values to be fed to shared computation units. These two counter examples show that high throughput LDPC codes decoder architecture should exploit the benefit of parallel property of the decoding algorithm while reducing the interconnection complexity.

Based on this observation the paper proposes an architecture that reconstructs the data flow of the iterative decoding to minimize the interconnection. While the previous implementations iteratively calculate messages that depends on each other, the proposed architecture separates the data flow into two paths by duplicating the computation units except row and column summation. Each path calculates one of row and column summation and completes the whole iteration path by referencing the summation of the other path. Furthermore, the messages are stored into segmented memory to reduce the area.

The rest of this paper is organized as follows. Section 2 summarizes low density parity check codes and their decoding algorithm. Section 3 proposes a new architecture for high throughput LDPC codes decoder based on segmented memory. An implementation example of the proposed architecture is presented in Section 4 along with its performance result, and Section 5 provides some concluding remarks.

II. Low Density Parity Check Codes

LDPC codes which were 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 a total number of n columns and J rows, where J represents the total number of parity check equations of the code. There are 2^K distinct codewords, where K is the message length, $K = n - J$. The numbers γ and ρ have to remain small with respect to n in order to obtain a sparse matrix H and they have are related to n and J as expressed in the following equation: $n \times \gamma = J \times \rho$. The code rate of LDPC codes is defined as $R = 1 - \gamma / \rho$. Figure 1 shows an example H matrix of (20, 3, 4) LDPC code.

A. Graph Representation

Kschischang et al. generalized the principles of Bayesian networks and Tanner graphs and developed a graphical model called "factor graph", which is a bipartite graph that expresses how a global function of many variables factors into a product

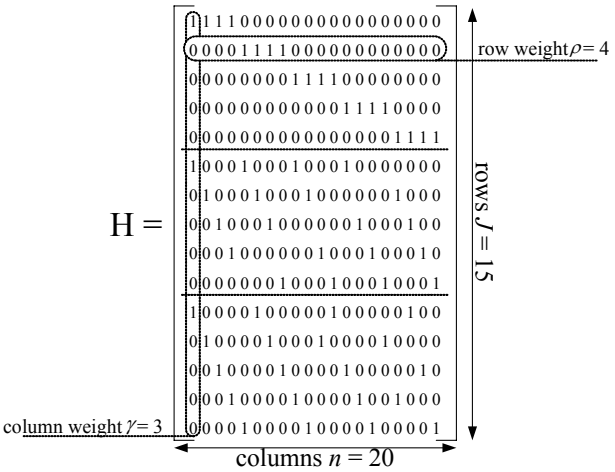

Figure 1 Example parity check matrix of (20, 3, 4) LDPC Code

of local functions [2]. In addition they showed how the a posteriori probability can be calculated by the probability propagation algorithm with a function of discrete variables [3]. Wiberg derived the decoding performance and analyzed the iterative decoding of the codes with the graph structure [4].

Figure 2 shows a factor graph for (7,4) Hamming code, which consists of two set of nodes, i.e. variable nodes, x_j and check nodes, S_i. Each row in the parity check matrix corresponds to variable nodes represented as circles and each column to check nodes represented as squares. The edges in the factor graph are constructed by the 1's in the H matrix. Given the factor graph, the message passing algorithm can be expressed. A posteriori probabilities for each bit in the codeword is calculated in the variable nodes and passed to check nodes through edges. Based on these probabilities, parity check operations are executed in each check nodes and the results are passed to variable nodes. Therefore fully parallel decoding architecture is inherently generated from the factor graph. Each variable and check node is implemented as a processing unit that generates the variable and check messages as many as the number of edges.

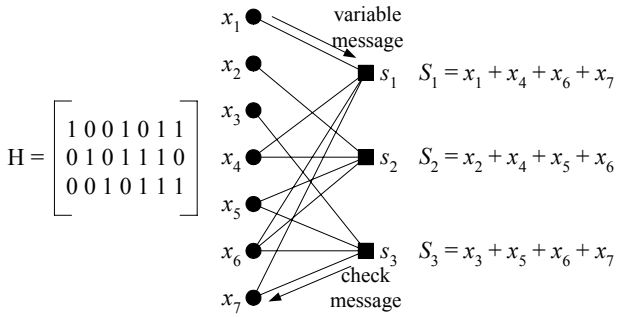

Figure 2 Factor graph for (7, 4) Hamming code

B. Motivation

Direct conversion of the message passing algorithm, however, faces with large storage requirement and complex interconnection because the number of message is twice as many as the number of edges and their interconnections are complex as shown in the factor graph. The proposed architecture aligns and concatenates these messages by row or column and stores them in segmented memories instead of mapping them to registers to alleviate the storage requirements. This is possible because the number of 1's in a row or column is small and small number of bits is sufficient to represent a message.

Employing memory, however, prevents simultaneous accesses to several elements in it. Therefore messages are grouped into as many as the number of parallel processing groups and stored into segmented memories. When grouping, the dependencies between row and column should be considered to minimize the processing latency. Detailed techniques on grouping and corresponding scheduling algorithm are described in the following section.

C. Decoding Algorithm

Unlike the general parity check codes, LDPC codes cannot be decoded optimally since it is NP-complete problem. Instead, an approximate algorithm called iterative probabilistic decoding is used, which is also known variously as sum-product [4] or belief propagation [12]. The sum-product algorithm which can be regarded as a message passing algorithm on a bipartite graph is described below.

1) Initialize a variable node to the probability ratio of the corresponding received bit.

2) Horizontal step: Compute the likelihood ratio corresponding to outgoing messages of variable nodes and propagate to check nodes.

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

3) Vertical step: From the variable messages, compute probability ratio corresponding to the outgoing check messages and propagate to variable nodes.

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

4) Create a tentative bit-by-bit decoding xi using pseudo-posteriori probability.

5) Check if $x \cdot H^T = 0$ is satisfied.

6) If the condition in step 5) is satisfied, the decoding algorithm finishes. Otherwise the algorithm repeats from the step 2).

The symbols, Δ and Λ, represent the outgoing messages of the check and the variable nodes, respectively, and $N(i)$, $M(j)$ represent the set of neighbor nodes connected by edges of check node i and variable node j, respectively. The not-notation(') which is marked as subscript in the indexes means that the product is calculated over the indexes excluding its own index. Detailed explanation of the algorithm including the equations can be found in [13]. To reduce the hardware cost, the products in the equations of the above algorithm description are usually transformed into summation by using log-likelihood probability. Therefore the resulting equations are as follows and the proposed architecture constructs two paths of data flow by combining these equations.

$$\Lambda_{ij} = \sum_{j' \in N(i) \setminus j} \log(\tanh(-\frac{\Delta_{ij'}}{2})) \qquad : \textit{horizontal step}$$

$$\Delta_{ij} = \log(\frac{p(r_j \mid x_j = +1)}{p(r_j \mid x_j = -1)}) - 2 \sum_{i' \in M(j) \setminus i} \tanh^{-1}(\exp(\Lambda_{i'j})) \qquad : \textit{vertical step}$$

III. Proposed LDPC Decoder Architecture

Since the functions of variable nodes and check nodes are simple, the main problem in the implementation is how to pass the large number of messages to other nodes. In the previous implementations, the problem is solved by providing complex interconnections. To tackle this problem, this paper proposes an architecture based on segmented memory to reduce overall area and enable high throughput decoding.

A. Duplicating the Data Flow

As shown in the Figure 5 (a), the previous data flow base on the message passing decoding generates considerable amount of messages from variable nodes to check nodes and vice versa which results in complex interconnection. Since Δ's in the same row are summed up to calculate their neighbor Λ's located at different columns and vice versa. Therefore complex interconnection cannot be avoided if the messages themselves are passed between nodes. The close look at the iterative decoding algorithm, however, tells that vertical and horizontal steps can be broken down to the summation of all the elements in the same row or column and subtraction of its own value from the summation. From this observation, this paper proposes an architecture which has two separate data flow by duplicating the processing elements except the summation.

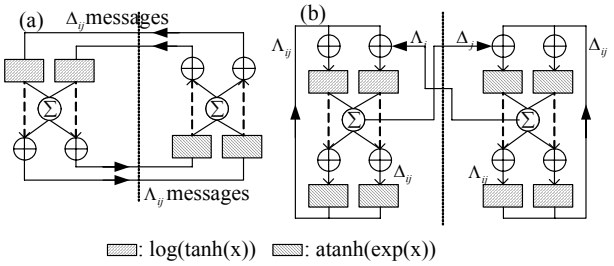

Figure 3 (a) Previous data flow (b) Duplicated data flow

In fact, complex interconnection is required to sum up the intermediate values in the same row or column which are aligned in the other way. To resolve this problem each data flow calculates only one of row and column summation value and passed it to other data flow. Since each data flow has the whole processing elements, the computation units for Δ and Λ memory calculate row and column summation respectively and simply these values, not the all messages, are fed into the other to construction complete iteration path. Therefore the resulting intermediate values can be written back to the same memory address where they are read for the next iteration. Thus index generation for writing results to each other's path is no longer necessary. This duplicated data flow is shown in the Figure 3 (b). As the whole row and columns are not processed simultaneously, duplicating processing units is not significant overhead. Precisely speaking, the intermediate values in the Δ and Λ memory is now no longer Δ and Λ define in the subsection 2.2.

B. Quantization and Segmentation

Due to the inherent robustness of Log-likelihood Ratio, Δ and Λ can be quantized into small number of bits. According to our experiment on the quantization effect against the number of bits in both integer and fractional parts, fractional part should be more emphasized than integer part and 5 bits which consists of 2 bit for integer part and 3 bits for fractional part are sufficient enough to express the Δ and Λ. Therefore Δ and Λ in the same row or column can be concatenated into one word and each row or column operation is enabled by just one memory read/write.

In addition, Δ and Λ memory is segmented to remove wasted cycles due to memory read and write. In other words, memory read and write at a row or a column occur while another row or column is used to calculate the summation value in the processing unit. Rows and columns which will be processed in parallel to achieve high throughput are also stored in different segments of memory.

C. Scheduling

If the intermediate values are grouped and stored into the segmented memory as described in the subsection 3.2, how the rows and columns are grouped and scheduled should be determined first. Since there are dependencies between each rows and columns, to find an optimum schedule that minimize latency due to the dependencies takes considerable amount of time. There may be heuristic algorithms and the authors propose an algorithm that reduces required rows and determine their order based on the given number of column groups are described below:

1) Make n group of columns which have common rows as many as possible.

2) Extract exclusive row list that is required to each column group

3) Sort each column group by the number of required rows

4) Put the group which requires smallest number of rows into a sequence.

5) Remove the rows of the just inserted column groups from the other column groups.

6) If the ordering is completed, the algorithm halts. Otherwise repeats from step 3).

Though the ordering can be determined for a given number of groups, the problem of how many columns are grouped into a group still remains. The number of groups can be determined after ordering for several numbers of groups are generated. For each case maximum required number of rows are varies not linearly proportional to number of groups, while

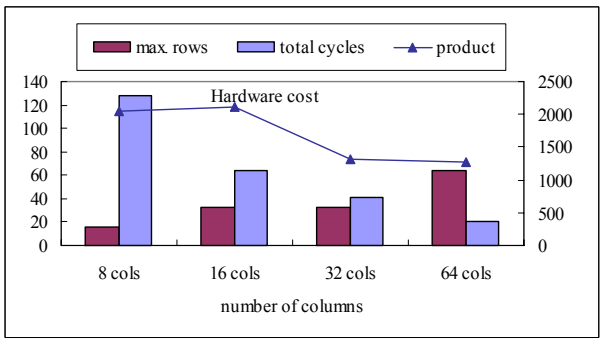

Figure 4 Processing time vs. hardware cost

the processing cycles are linearly proportional. Therefore some trade-offs can be done between area and processing time. Figure 4 shows some experimental result of ordering for several numbers of groups. The bar in the graph represents maximum number of rows which corresponds the hardware cost and total cycles for one iteration. The break line represents the product of these two values as a measure of trade-off between throughput and hardware cost.

IV. Implementation and Performance

We design a (1024, 3, 6) LDPC code decoder using the proposed architecture. The block diagram of our implementation is plotted in Figure 5. The overall architecture is divided into two paths according to Δ and Λ memory. In each path there are processing elements which consists of lookup tables for the functions, log(tanh(x)) and atanh(exp(x)), and adders for each segmented memory. And there is array of D F/F for easy accessing of row and column summation values and multiplexers to feed them to corresponding processing elements at the other side. Simply incrementing the address of segmented memory, processing elements read and write the intermediate values to calculate the summation. Column summation values are bit-by-bit decoded according to their sign bits and become final output.

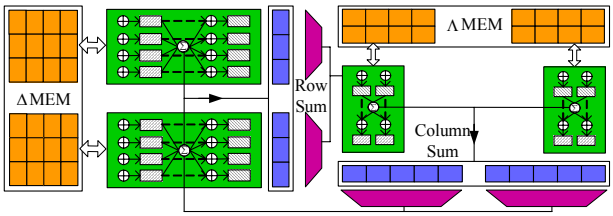

Figure 5 Block diagram of the proposed architecture

The performance of the proposed architecture with 64 parallel processing for (1024, 3, 6) LDPC codes is summarized in Table 1. Bit rate is comparable to the fully parallel architecture implemented by Blanksby et al., but significant area reduction is achieved. While the number of iterations is determined to 64 from the architecture of Blanksby's implementation, there is no restriction in the proposed architecture and thus we can choose 8 iterations. Most of the area of the proposed architecture is occupied by the Δ and Λ memory, and this fact proves that the duplicating the processing elements is a little overhead. The case when the intermediate values are store in the D F/F's instead of segmented memory is also shown in the Table 1. This improves bit rate a little by removing a few cycle of memory read/write latencies but the area overhead is considerable.

Table 1 Performance summary

	Blanksby [9]	Segmented Memory	D F/F
Technology	0.16 um	0.25 um	0.25 um
Bit rate	1 Gbps	853 Mbps	960 Mbps
Area	52.5 mm^2	4.75 mm^2	8.71 mm^2
		MEM: 3.05 mm^2	6.12 mm^2
		Other: 1.78 mm^2	2.59 mm^2

V. Conclusion

Large storage elements and complex interconnection are the main obstacles in VLSI implementation of the LDPC codes decoders while their decoding complexity is relatively low. This paper has proposed a memory segmentation technique and a corresponding scheduling algorithm for memory-based LDPC codes decoding architecture. Compared to the fully parallel architecture, the proposed architecture saves the overall area significantly and simplifies interconnection by storing intermediate values (Δ and Λ) into segmented memory associated with rows and columns. Since the overall structure of the proposed architecture is highly regular, it is easy to scale up and down the parallelism considering a given design constraints.

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, pp. 533-547, Jan. 1962.

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

[3] F. R. Kschischang, "Iterative decoding of compound codes by probability propagation in graphical models," IEEE J. on Selected Areas in Communications, pp. 219-230, Feb. 1998.

[4] N. Wiberg, Codes and Decoding on General Graphs, PhD thesis, Dept. of EE, Linköping Univ. Sweden, 1996.

[5] D. J. C. Mackay, "Good error-correcting codes based on very sparse matrices," IEEE Trans. Info. Theory, pp. 399-431, Mar. 1999.

[6] 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, pp. 58-60, Feb. 2001.

[7] Andrew J. Blanksby and Chris J. Howland, "A 690-mW 1-Gb/s, Rate-1/2 low-density parity-check code decoder," IEEE J. of Solid-State Circuits, pp. 404-412, Mar. 2002.

[8] Engling Yeo, Payam Pakzad, Borivoje Nikolić and Venkat Anantharam, "VLSI Architectures for Iterative Decoders in Magnetic Recording Channels," IEEE Trans. Magnetics, pp. 748-755, Mar. 2001.

[9] E. Eleftheriou and S. Ölçer, "Low-density parity-check codes for digital subscriber lines," IEEE International Conference on Communications, pp. 1752-1757, May 2002.

[10] Vladislav Sorokine, Frank R. Kschischang and Subbarayan Pasupathy, "Gallager codes for CDMA applications – Part I: Generalizations, constructions and performance bounds," IEEE Trans. Communications, pp. 1660-1668, Oct. 2000.

[11] Ben Lu, Xiaodong Wang and Krishna R. Narayanan, "LDPC-based space-time coded OFDM systems over correlated fading channels: Performance analysis and receiver design," IEEE Trans. Communications, pp. 74-88, Jan. 2002.

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

[13] Ka Leong, Lo, Layered Space Time Structures with Low Density Parity Check and Convolutional Codes, MS Thesis, School of EIE, Univ. of Sydney, 2001.