Parallel LDPC Decoder Based on Low-Complexity Corrected Min Sum Algorithm

Yisong Sun
University of Chinese Academy of Sciences (UCAS)
Institute of Automation, Chinese Academy of Sciences (CASIA)
Beijing, China
sunyisong2016@ia.ac.cn

Huan Li
Institute of Automation, Chinese Academy of Sciences (CASIA)
Chinese Academy of Sciences (CAS)
Beijing, China
lihuan2013@ia.ac.cn

Xinyu Zhang
Department of Applied Algorithm
Beijing Smart Logic East Technology Co., Ltd.
Beijing, China
xinyu.zhang@smartlogictech.com

Chen Guo
Institute of Automation, Chinese Academy of Sciences (CASIA)
Chinese Academy of Sciences (CAS)
Beijing, China
chen.guo@ia.ac.cn

Zijun Liu
Institute of Automation, Chinese Academy of Sciences (CASIA)
Chinese Academy of Sciences (CAS)
Beijing, China
zijun.liu@ia.ac.cn

Donglin Wang
Institute of Automation, Chinese Academy of Sciences (CASIA)
Chinese Academy of Sciences (CAS)
Beijing, China
donglin.wang@ia.ac.cn

***Abstract*—To correct the errors introduced by the approximation in the Min Sum (MS) algorithm with low complexity, we proposed Low-Complexity Corrected Min Sum (LCC-MS) algorithm. Aiming at the implementation bottleneck of the algorithm on the self-developed DSP (Universal Communication Processor, UCP), we optimized the algorithm process and expanded the instruction set of UCP. Therefore, the LDPC decoder based on LCC-MS algorithm is implemented on UCP, and verified on the chip that has been taped out. The verification results of the longest code length and the highest code rate show that the LCC-MS algorithm has a gain of 0.16dB when the bit error rate is 10^{-5} compared with the MS algorithm, and the decoder based on LCC-MS algorithm can process up to 149 code blocks in one time slot.**

Keywords—5G, LDPC decoder, LCC-MS, UCP

I. Introduction

Since the era of 2G communication, the channel coding and decoding technology has been one of the key technologies of the physical layer of the wireless communication system. An excellent channel coding and decoding scheme can provide greater data throughput, better transmission quality, lower transmission delay and lower energy consumption for wireless communication [1]. Among many channel coding and decoding technologies, Low Density Parity Check (LDPC) code is a block error-correction code with sparse parity check matrix proposed by Robert Gallager of MIT in his doctoral thesis [2] in 1962. LDPC code is suitable for almost all channels, and its performance is close to the Shannon limit. The decoding process of LDPC code is simple and can be implemented in parallel, which is suitable for hardware implementation. At the 87th meeting of 3GPP RAN1 held in Las Vegas in November 2016, LDPC code was selected as the error control coding scheme for data channel of 5G communication system [3]. Therefore, the research on LDPC encoding and decoding technology is of great significance for breaking through the performance bottleneck of 5G communication systems.

Regarding the implementation of LDPC code decoders, researchers in related fields have carried out a lot of researches. Mouhcine Razi et al. [4] use Horizontal Shuffled scheduling instead of the usually used flooding scheduling and implement it on the Digital Signal Processor. The results show that the proposed algorithm not only improves the decoding performance but also decreases the number of iterations. Aleksei Kharin et al. [5] implement an LDPC belief propagation decoder on a DSP+ARM platform. The effectiveness of the platform is demonstrated, with up to 2 decoding iterations executing in real time. Mouhcine Razi et al. [6] introduce two new improved versions of the hard-decision algorithms, the adaptative gradient descent bit-flipping and adaptative reliability ratio weighted gradient descent bit-flipping. The results of numerical simulations and DSP implementation reveal a faster convergence with a low processing time and a reduction in consumed memory resources. Gottfried Lechner et al. [7] present a high-performance implementation of a belief propagation decoder for decoding low-density parity-check codes on a fixed-point digital signal processor. The decoder can decode at 5.4Mbps on a Texas Instruments TMS320C64xx DSP running at 600MHz.

Literature survey shows that the current decoding algorithm researches are mostly based on Belief Propagation (BP) algorithm, also known as Sum Product (SP) algorithm. The Min Sum (MS) algorithm greatly reduces the computational complexity of the SP algorithm at the cost of introducing errors. For the errors introduced by the MS algorithm, the existing researches have not proposed a correction scheme with lower complexity, so that the decoding algorithm cannot take into

account the improvement of the data throughput performance and the bit error rate performance.

To solve this problem, we proposed an improved algorithm and implemented an LDPC decoder based on the algorithm on our self-developed DSP (Universal Communication Processor, UCP). Compared with other existing researches, the main contributions of our works are:

- We proposed Low-Complexity Corrected Min Sum (LCC-MS) algorithm to correct the errors introduced by the approximation in the MS algorithm with low complexity, to achieve better bit error rate performance while ensuring data throughput.
- By considering the architectural characteristics of UCP, we optimized the algorithm process and expanded the instruction set of UCP, effectively solving the performance bottleneck of the algorithm on UCP.

The rest of this paper is organized as follows: Chapter 2 introduces the basic principles of the MS algorithm. Chapter 3 presents the design of the LCC-MS algorithm. Chapter 4 and Chapter 5 introduce the hardware implementation of the check node and variable node, respectively. Chapter 6 and Chapter 7 give the experimental results and conclusions, respectively.

II. PRELIMINARIES

The concept of LDPC code and its iterative decoding algorithm can be traced back to the early 1960s. Gallager defined regular LDPC codes in his doctoral thesis [2], so named because of the small number of non-zero elements in the parity check matrix. In 1981, Tanner proposed the bidirectional graph representation of LDPC codes [8], also known as Tanner graphs, which laid the foundation for the decoding algorithm of LDPC codes. In 1997, MacKay et al. proposed the Sum Product (SP) algorithm [9], also known as the Belief Propagation (BP) algorithm. The SP algorithm is an optimal LDPC decoding algorithm based on iterative decoding. This algorithm mainly consists of two steps: the update of the check node and the update of the variable node. To avoid a large number of multiplication operations, the decoding process is often performed in the log-likelihood domain in practical applications. The SP algorithm can be further optimized, such as the Min Sum (MS) decoding algorithm [10] that simplifies the check node processing.

The MS algorithm includes three steps: initialization, check node processing, and variable node processing.

A. Initialization

We denote the output sequence of the LDPC encoder as x_n, the symbol sequence received by the receiver as y_k, the prior Log-Likelihood Ratio (LLR) of variable node n derived from y_k as F_n, and the LLR of variable node n passed to check node m as Z_{mn}. This step assigns an initial value to each variable node, that is, the prior LLR:

$$Z_{mn} = F_n = \log_2 \frac{P\left(x_n = 0 \mid y_k\right)}{P\left(x_n = 1 \mid y_k\right)} \tag{1}$$

B. Check node processing

We denote the hard decision result of Z_{mn} as σ_{mn}, the result of modulo-2 addition of σ_{mn} related to check node m as σ_m, and the LLR of check node m passed to variable node n as L_{mn}. First calculates σ_{mn} and σ_m according to Z_{mn}:

$$\sigma_{mn} = sign\left(Z_{mn}\right) \tag{2}$$

$$\sigma_m = \sum_{n \in N(m)} \sigma_{mn} \bmod 2 \tag{3}$$

The sign function in this paper means to get the sign bit in the complement representation. For each check node m, compute the LLR that it passes to each variable node connected to itself:

$$\begin{aligned} L_{mn} &= \ln \frac{1 + \prod_{n' \in N(m) \backslash n} \frac{e^{z_{mn'}} - 1}{e^{z_{mn'}} + 1}}{1 - \prod_{n' \in N(m) \backslash n} \frac{e^{z_{mn'}} - 1}{e^{z_{mn'}} + 1}} \\ &= \ln \frac{1 + \prod_{n' \in N(m) \backslash n} \tanh \frac{Z_{mn'}}{2}}{1 - \prod_{n' \in N(m) \backslash n} \tanh \frac{Z_{mn'}}{2}} \\ &= 2 \tanh^{-1} \left(\prod_{n' \in N(m) \backslash n} \tanh \frac{Z_{mn'}}{2} \right) \end{aligned} \tag{4}$$

From the properties of the hyperbolic tangent function, the absolute value of $\tanh \frac{Z_{mn'}}{2}$ is very close to 1 in most cases, so we can only keep the item with the smallest absolute value among the consecutive multiplication items, approximate the absolute value of the other items as 1, and only keep the signs of the other items:

$$\begin{aligned} L_{mn} &\approx 2 \tanh^{-1} \left(\left(-1\right)^{\sigma_m \oplus \sigma_{mn}} \min_{n' \in N(m) \backslash n} \left| \tanh \frac{Z_{mn'}}{2} \right| \right) \\ &= 2 \tanh^{-1} \left(\left(-1\right)^{\sigma_m \oplus \sigma_{mn}} \tanh \frac{\min_{n' \in N(m) \backslash n} \left| Z_{mn'} \right|}{2} \right) \\ &= \left(-1\right)^{\sigma_m \oplus \sigma_{mn}} \min_{n' \in N(m) \backslash n} \left| Z_{mn'} \right| \end{aligned} \tag{5}$$

C. Variable node processing

We donate the LLR of check node m passed to variable node n as L_{mn}, the posterior LLR of variable node n as Z_n, and the hard decision result of Z_n as σ_n. For each variable node n, compute the LLR that it passes to each check node m connected to itself:

$$Z_{mn} = F_n + \sum_{m' \in M(n) \backslash m} L_{m'n} \tag{6}$$

For each variable node n, compute its posterior LLR:

$$Z_n = F_n + \sum_{m \in M(n)} L_{mn} \tag{7}$$

The bit sequence σ_n is obtained by hard decision based on posterior LLR:

$$\sigma_n = sign(Z_n) \tag{8}$$

The bit sequence σ_n is checked with the check matrix, and if the check is passed, the decoding result is output and the decoding ends. Otherwise, repeat the above iterative process until the maximum number of iterations is reached.

III. Low-Complexity Corrected Min Sum Algorithm

For the error introduced by the approximation in the MS algorithm, we propose the Low-Complexity Corrected Min Sum (LCC-MS) algorithm to correct the error with low complexity. It can be seen from (5) that the absolute value of other terms is approximated to 1 when L_{mn} is calculated. For the error introduced by this approximation, we determine a correction factor α through simulation to correct the error:

$$\begin{aligned} L_{mn} &\approx 2\alpha \tanh^{-1}\left((-1)^{\sigma_m \oplus \sigma_{mn}} \min_{n' \in N(m)\backslash n} \left| \tanh \frac{Z_{mn'}}{2} \right| \right) \\ &= 2\alpha \tanh^{-1}\left((-1)^{\sigma_m \oplus \sigma_{mn}} \tanh \frac{\min_{n' \in N(m)\backslash n} |Z_{mn'}|}{2} \right) \\ &= (-1)^{\sigma_m \oplus \sigma_{mn}} \alpha \min_{n' \in N(m)\backslash n} |Z_{mn'}| \end{aligned} \tag{9}$$

We denote the current iteration number as p, the number of check nodes as M_0, the number of variable nodes as N_0, the average number of check nodes connected to each variable node as M_1, the average number of variable nodes connected to each check node as N_1, and the number of correction factor multiplications required for each iteration as S_1, then

$$S_1 = N_0 M_1 \tag{10}$$

Since the LLR transmitted in the iterative process is used for absolute value comparison and hard decision, multiplying the LLR by a constant greater than 0 does not affect the decoding result. So we can use $\frac{L_{mn}}{\alpha^p}$ instead of L_{mn} and $\frac{Z_{mn}}{\alpha^p}$ instead of Z_{mn} to participate in the iterative process. As a result, we can change (5) to

$$\frac{L_{mn}}{\alpha^p} \approx (-1)^{\sigma_m \oplus \sigma_{mn}} \min_{n' \in N(m)\backslash n} \left| \frac{Z_{mn'}}{\alpha^{p-1}} \right| \tag{11}$$

change (6) to

$$\frac{Z_{mn}}{\alpha^p} = \frac{F_n}{\alpha^p} + \sum_{m' \in M(n)\backslash m} \frac{L_{m'n}}{\alpha^p} \tag{12}$$

change (7) to

$$\frac{Z_n}{\alpha^p} = \frac{F_n}{\alpha^p} + \sum_{m \in M(n)} \frac{L_{mn}}{\alpha^p} \tag{13}$$

In this case, it is only necessary to compute $\frac{F_n}{\alpha^p}$ for each variable node per iteration. Since α is a constant, each power of α can be pre-computed and stored, and read directly when needed. The number of correction factor multiplications required for each iteration after optimization is

$$S_1 = N_0 \tag{14}$$

Compared with (10), it can be seen that the LCC-MS algorithm reduces the number of correction factor multiplications from $O(N_0 M_1)$ to $O(N_0)$, thus correcting the error introduced in the MS algorithm with a lower complexity.

To implement the LDPC decoder based on the LCC-MS algorithm on our self-developed Universal Communication Processor (UCP), we have improved the LCC-MS algorithm flow under the premise of considering the characteristics of UCP. This reduces the overall computational load and improves the degree of parallelism, making the algorithm more suitable for implementation on UCP. There are two aspects of optimization, which will be discussed below.

A. *Optimization of minimum absolute value solving in complement set*

It can be seen from (11) that each time $\frac{L_{mn}}{\alpha^p}$ is calculated, the minimum absolute value of the incoming LLRs needs to be found in the complement set. Denote the number of comparison operations required for each iteration as S_2, then

$$S_2 = M_0 N_1 (N_1 - 1) \tag{15}$$

It can be observed that the minimum absolute value of the incoming LLRs in the complement set is at most the second minimum absolute value in the incoming LLRs in the complete set, that is,

$$\min_{n' \in N(m)\backslash n} \left| \frac{Z_{mn'}}{\alpha^{p-1}} \right| = \begin{cases} \min_{n' \in N(m)} \left| \frac{Z_{mn'}}{\alpha^{p-1}} \right|, \text{ if } \left| \frac{Z_{mn}}{\alpha^{p-1}} \right| \neq \min_{n' \in N(m)} \left| \frac{Z_{mn'}}{\alpha^{p-1}} \right| \\ \text{second} \min_{n' \in N(m)} \left| \frac{Z_{mn'}}{\alpha^{p-1}} \right|, \text{ if } \left| \frac{Z_{mn}}{\alpha^{p-1}} \right| = \min_{n' \in N(m)} \left| \frac{Z_{mn'}}{\alpha^{p-1}} \right| \end{cases} \tag{16}$$

Therefore, we can compare the incoming LLRs in the complete set N_1 times to find the minimum absolute value and the second minimum absolute value. Then, for each edge, judge whether the absolute value of the incoming LLR is equal to the minimum absolute value of the incoming LLRs in the complete set, and obtain the minimum absolute value of the incoming LLR in the complement set accordingly. The number of comparison operations required for each iteration after optimization is

$$S_2 = M_0 (N_1 + N_1) = 2 M_0 N_1 \tag{17}$$

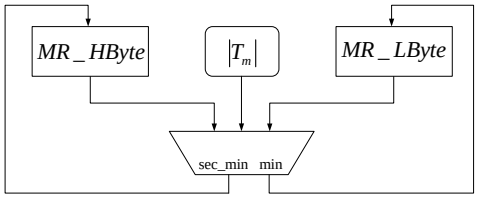

Fig. 1. Fuction description of CompMR instruction. The multiplexer takes the absolute value of T_m, the low byte of MR and the high byte of MR as input, and selects the minimum value and the second minimum value and stores them back into the low byte and high byte of MR respectively.

Compared with (15), it can be seen that the optimization reduces the number of comparison operations from $O\left(M_0 N_1^2\right)$ to $O\left(M_0 N_1\right)$.

B. Optimization of variable node processing

It can be seen from (12) that each time $\frac{Z_{mn}}{\alpha^p}$ is calculated, the incoming LLRs in the complement set need to be accumulated. Denote the number of accumulation operations required for each iteration as S_3, then

$$S_3 = N_0 M_1 \left(M_1 - 1\right) \tag{18}$$

We can first calculate $\frac{Z_n}{\alpha^p}$ according to (13), and then calculate $\frac{Z_{mn}}{\alpha^p}$ according to $\frac{Z_n}{\alpha^p}$, that is, change (12) to

$$\frac{Z_{mn}}{\alpha^p} = \frac{Z_n}{\alpha^p} - \frac{L_{mn}}{\alpha^p} \tag{19}$$

The number of accumulation operations required for each iteration after optimization is

$$S_3 = N_0 \left(M_1 + M_1\right) = 2 N_0 M_1 \tag{20}$$

Comparing with (18), this optimization can reduce the number of accumulation operations from $O\left(N_0 M_1^2\right)$ to $O\left(N_0 M_1\right)$.

IV. Hardware Implementation of Check Node Processing

This chapter will introduce the specific implementation of key steps in check node processing, as well as the design of the overall implementation scheme of check node processing. The key steps include the solving of the minimum absolute value and the second minimum absolute value of the incoming LLRs in the complete set, the solving of the minimum absolute value of the incoming LLRs in the complement set and the appending of the symbolic term.

It should be noted that, for the sake of brevity, most of the figures in this paper only show the processing of a single byte. In fact, the instructions mentioned in this paper can all process multiple bytes in parallel with the specified degree of parallelism. This will not be repeated in the description of each figure below.

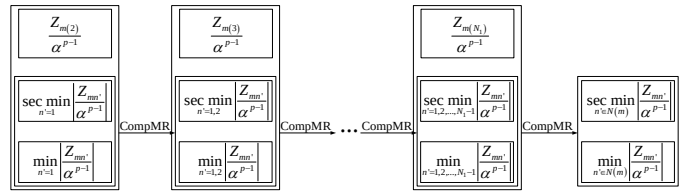

Fig. 2. Process of iteratively solving the minimum absolute value and the second minimum absolute value in the complete set. Traverse each incoming LLR in the complete set, and iteratively update the current minimum absolute value and the second minimum absolute value. After the traversal is completed, the minimum absolute value and the second minimum absolute value in the complete set are obtained.

A. Solving of minimum and second minimum absolute value in complete set

In this step, it is necessary to traverse the incoming LLRs of the N_1 edges connected to the current check node to find the minimum absolute value and the second minimum absolute value. It takes a lot of computing resources to complete this operation through conventional instructions. Therefore, we customized the absolute value comparison instruction CompMR based on the UCP architecture. The instruction function is described in Fig. 1.

We use this instruction to solve the minimum absolute value and the second minimum absolute value in the complete set, as shown in Fig. 2.

B. Solving of minimum absolute value in complement set

This step needs to determine whether the absolute value of the incoming LLR is equal to the minimum absolute value obtained in the previous step for each edge connected to the current check node. If not equal, select the minimum absolute value as the output result, otherwise select the second minimum absolute value as the output result. We have customized the absolute value comparison selection instruction CompAbsSel based on the UCP architecture. The instruction function is described in Fig. 3.

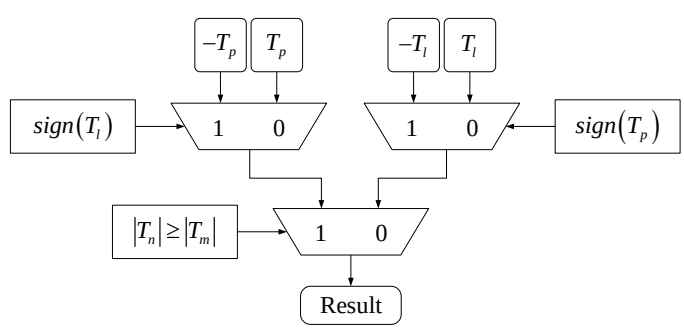

Fig. 3. Fuction description of CompAbsSel instruction. This instruction is formed by cascading two stages of multiplexers. The first-stage multiplexers select to output T_p or $-T_p$ through the sign bit of T_l, and select to output T_l or $-T_l$ through the sign bit of T_p. The second-stage multiplexer selects which first-stage multiplexer to output through the comparison result of the absolute value of T_n and T_m.

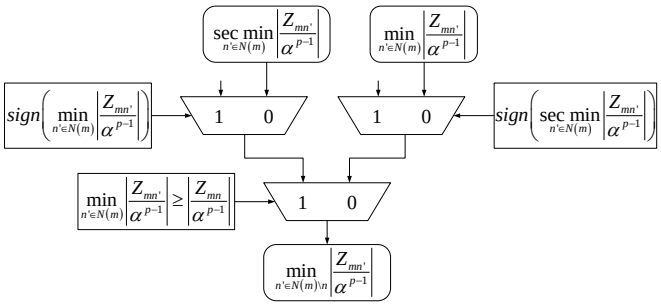

Fig. 4. Process of solving the minimum absolute value in the complement set. Since the absolut values must be non-negative, the first-stage multiplexer only selects the input of 0. The second-stage multiplexer is based on the comparison expression between the minimum value and current value, which is true if and only if the two are equal since the minimum value cannot be greater than current value. In this situation, the second minimum value is selected as the output. Otherwise select the minimum value as the output. The output conforms to the minimum absolute value in the complement set.

The function of this instruction can also be described by the following expression:

$$|T_n| \geq |T_m| ?\left(sign\left(T_l\right)?-T_p : T_p\right) : \left(sign\left(T_p\right)?-T_l : T_l\right) \quad (21)$$

We use this instruction to solve the minimum absolute value of the incoming LLRs in the complement set, and configure T_m as $\left|\frac{Z_{mn}}{\alpha^{p-1}}\right|$, T_n and T_l as $\min\limits_{n'\in N(m)}\left|\frac{Z_{mn'}}{\alpha^{p-1}}\right|$, and T_p as $\text{second}\min\limits_{n'\in N(m)}\left|\frac{Z_{mn'}}{\alpha^{p-1}}\right|$, as shown in Fig.4.

C. Appending of symbolic term

This step needs to append the symbolic term $(-1)^{\sigma_m \oplus \sigma_{mn}}$ in (11) to the output of the previous step. Using the traditional multiplication instruction will consume a large amount of computing resources, so we reuse the customized CompAbsSel instruction in the previous step, and configure T_m as 0, T_n as $\left|\frac{Z_{mn}}{\alpha^{p-1}}\right|$, and T_p as $\min\limits_{n'\in N(m)\backslash n}\left|\frac{Z_{mn'}}{\alpha^{p-1}}\right|$ to realize the appending of symbolic item, as shown in Fig. 5.

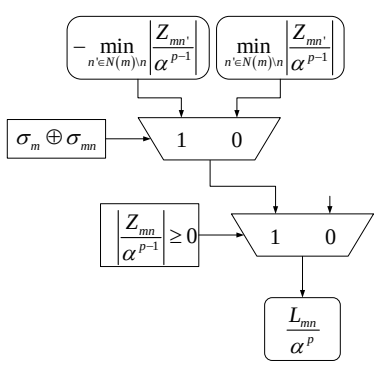

Fig. 5. Process of symbolic term appending. Since the absolut values must be non-negative, the second-stage multiplexer only selects the input of 1 in this step. The first-stage multiplexer selects to output minimum absolute value in the complement set or its negative value through $\sigma_m \oplus \sigma_{mn}$, thereby realizing the appending of symbolic term.

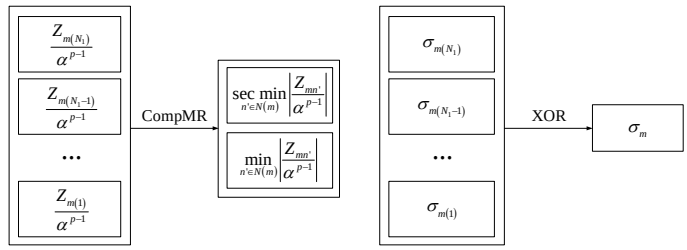

Fig. 6. Pre-computation stage of check node process. Minimum absolute value, second minimum absolute value and σ_m are obtained at this stage.

D. Overall scheme design

The processing of the check node can be divided into two stages. Pre-computation stage: traverse the incoming LLRs of all edges through the CompMR instruction to obtain the minimum absolute value and the second minimum absolute value, and perform modulo-2 addition on the sign bits of the incoming LLRs of all edges through the XOR logic instruction, as shown in Fig. 6.

Parallel processing stage: For each edge, the minimum absolute value of the incoming LLRs in the complement set is solved in parallel by the CompAbsSel instruction, the symbolic term is solved in parallel by the XOR logic instruction, and the symbolic term is appended to the minimum absolute value of the incoming LLRs in the complement set in parallel by the CompAbsSel instruction, thereby obtaining the outgoing LLR of each edge in parallel, as shown in Fig. 7.

V. Hardware Implementation of Variable Node Processing

This chapter will introduce the specific implementation of key steps in variable node processing, as well as the design of the overall implementation scheme. The key step is the hard decision for posterior LLR of variable nodes.

A. Hard decision for posterior LLR of variable nodes

This step requires a hard decision based on the sign of $\frac{Z_n}{\alpha^p}$. If $\frac{Z_n}{\alpha^p}$ is positive, the decision is 0, otherwise the decision is 1. We use the comparison selection instruction CompSel to implement this decision. The instruction function is described in Fig. 8 (a) and (22).

$$T_n \geq T_m ?T_p : T_l \quad (22)$$

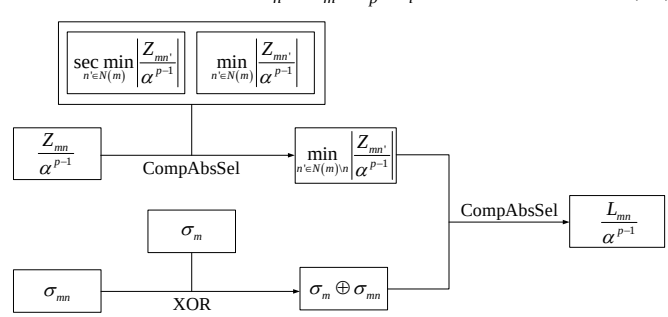

Fig. 7. Parallel processing stage of check node process. The outgoing LLR passed to variable node is obtained at this stage.

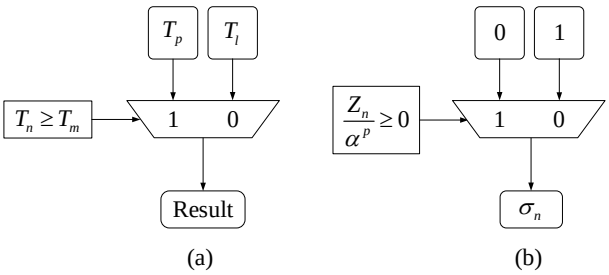

Fig. 8. Fuction description of CompSel instruction is shown in (a). The multiplexer selects T_p or T_p according to the expression $T_n \geq T_m$. Hard decision process is shown in (b). Outputs 0 when the expression is true, and 1 otherwise. This is in line with the definition of hard decision.

We configure T_n as $\frac{Z_n}{\alpha^p}$, T_m and T_p as 0, and T_l as 1 to implement the hard decision of $\frac{Z_n}{\alpha^p}$, as shown in Fig. 8 (b).

B. Overall scheme design

The processing of variable nodes can be divided into two stages. Pre-computation stage: $\frac{F_n}{\alpha^p}$ is obtained by multiplication instruction, and $\frac{Z_n}{\alpha^p}$ is obtained by accumulating $\frac{L_{mn}}{\alpha^p}$ by addition instruction, and a hard decision is made on $\frac{Z_n}{\alpha^p}$ through CompSel instruction, as shown in Fig. 9 (a).

Parallel processing stage: For each edge, the outgoing LLR is solved in parallel by the subtraction instruction according to (19), as shown in Fig. 9 (b).

VI. Experimental Results and Discussion

To evaluate the performance of the LDPC decoder based on the LCC-MS algorithm, we implemented the encoder with UCP's self-developed assembly language, and performed RTL simulation with UCP's own tool chain, and performed hardware verification on the tape-out UCP chip. The UCP chip adopts Taiwan Semiconductor Manufacturing Company (TSMC) 16-nm tape-out process, which can work stably at 1.2GHz main frequency.

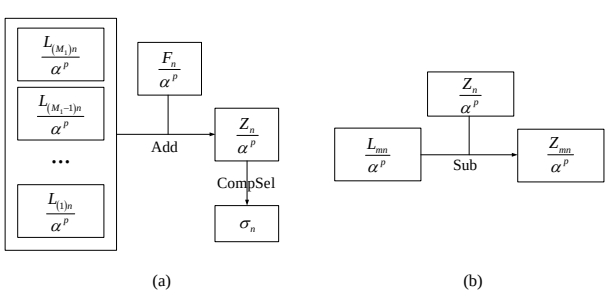

Fig. 9. Pre-computation stage of variable node process is shown in (a). Parallel processing stage of variable node process is shown in (b).

TABLE I. Parameters and Performance

Parameters		Performance	
Information bits length (bits)	8448	Processing time (μs)	3.34
Code rate	0.92	Data throughput (Gbps)	2.53
Expansion factor (bits)	384	Number of cycles	4013

We selected the maximum code length and maximum code rate agreed in the 5G protocol [3] for performance verification, as shown in Table I. First, we compare the bit error rate performance of the LCC-MS algorithm and the MS algorithm through floating-point simulation. As shown in Fig. 10, when the bit error rate is 10^{-5}, the LCC-MS algorithm obtains a gain of about 0.16dB compared to the MS algorithm. This proves that the LCC-MS algorithm can effectively correct the errors introduced in the MS algorithm. Then we converted the LCC-MS algorithm into fixed-point form, and implemented the algorithm on the chip with 8-bit fixed-point data accuracy. As shown in Fig. 10, when the bit error rate is 10^{-5}, the fixed-point algorithm only loses about 0.04dB compared to the floating-point algorithm.

At the same time, we test the data throughput performance of the decoder based on the LCC-MS algorithm on the chip. As shown Table I, the processing time for a single code block is 3.34μs. Therefore, within a time slot of 0.5ms, the decoder can process up to 149 code blocks, which is significantly more than the maximum number of code blocks agreed in the 5G protocol [3]. This proves that the LCC-MS algorithm can correct the error while ensuring the decoder achieves high data throughput.

To sum up, the LDPC decoder based on the LCC-MS algorithm corrects the errors introduced by the MS algorithm with a lower complexity, reduces the bit error rate on the premise of ensuring a higher data throughput rate, thereby improving the overall performance of the communication link. According to our analysis, its excellent performance is mainly due to the following two aspects:

First, the excellent performance benefits from the design and process improvements of the LCC-MS algorithm. The design of

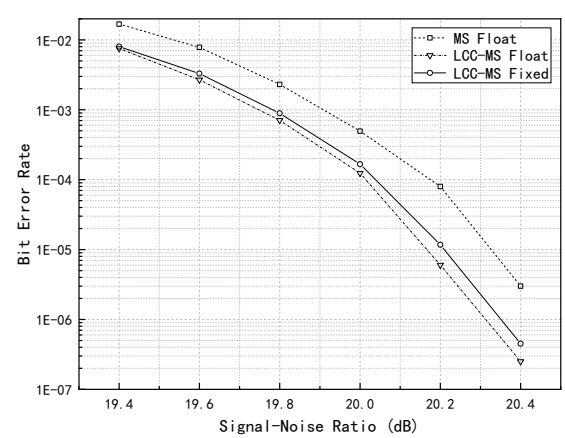

Fig. 10. Bit error rate curves. There are three curves in the figure, which are the bit error rate curve of floating-point MS algorithm, the bit error rate curve of floating-point LCC-MS algorithm and the bit error rate curve of fixed-point LCC-MS algorithm.

the LCC-MS algorithm reduces the number of correction factor multiplications from $O(N_0 M_1)$ to $O(N_0)$. In addition, the improvement of the process for the hardware architecture also significantly reduces the number of comparison operations and accumulation operations. The performance speedup is especially significant when the number of nodes is large.

Second, the excellent performance benefits from the architectural advantages of UCP. UCP's powerful computing power and high-speed main frequency are the underlying guarantee for excellent performance. UCP's efficient parallel architecture allows the computation of the algorithm to be executed in parallel. UCP's easy-to-expand instruction set allows us to easily customize instructions, such as CompAbsSel instruction, thus solving the key implementation bottleneck of the algorithm.

VII. Conclusions

In this paper, we proposed the LCC-MS algorithm to correct the error introduced by the MS algorithm with lower complexity. Aiming at the implementation bottleneck of the algorithm in hardware, we optimized the algorithm flow and expanded the instruction set of UCP. Thus, the LCC-MS-based LDPC decoder is implemented on UCP, and the performance is verified on the tape-out chip. The experimental results show that the LCC-MS algorithm has a gain of 0.16dB when the bit error rate is 10^{-5} compared with the MS algorithm, and the decoder based on LCC-MS can process up to 149 code blocks in one time slot. Therefore, we believe that the LCC-MS-based decoder is able to correct the errors introduced in the MS algorithm with lower complexity.

Although we only evaluate the performance of the LCC-MS-based LDPC decoder on UCP, the decoder design is equally applicable to other SIMD architectures. In future researches, we will continue to explore parallelization schemes for other physical layer algorithms of 5G communication, and conduct researches on efficient hardware implementation on UCP.

References

[1] Hui, Dennis, et al. "Channel coding in 5G new radio: A tutorial overview and performance comparison with 4G LTE." ieee vehicular technology magazine 13.4 (2018): 60-69.

[2] Gallager, Robert. "Low-density parity-check codes." IRE Transactions on information theory 8.1 (1962): 21-28.

[3] 3GPP TS 38.212. "NR; Multiplexing and channel coding." (2020).

[4] Razi, Mouhcine, et al. "Fast DSP Implementation of a Low Complexity LDPC Decoder." 2019 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS). IEEE, 2019.

[5] Kharin, Aleksei, et al. "LDPC decoder implementation on DSP+ ARM platform with OpenCL." 2018 7th Mediterranean Conference on Embedded Computing (MECO). IEEE, 2018.

[6] Razi, Mouhcine, et al. "An improvement and a fast DSP implementation of the bit flipping algorithms for low density parity check decoder." International Journal of Electrical & Computer Engineering (2088-8708) 11.6 (2021).

[7] Lechner, Gottfried, Jossy Sayir, and Markus Rupp. "Efficient DSP implementation of an LDPC decoder." 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing. Vol. 4. IEEE, 2004.

[8] Tanner, R. "A recursive approach to low complexity codes." IEEE Transactions on information theory 27.5 (1981): 533-547.

[9] MacKay, David JC, and Radford M. Neal. "Near Shannon limit performance of low density parity check codes." Electronics letters 32.18 (1996): 1645.

[10] Fossorier, Marc PC, Miodrag Mihaljevic, and Hideki Imai. "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation." IEEE Transactions on communications 47.5 (1999): 673-680.