A High-Throughput LDPC Decoder Based on GPUs for 5G New Radio

Rongchun Li*, Xin Zhou*, Hengyue Pan, Huayou Su, Yong Dou
National Key Laboratory for Parallel and Distribution Processing
National University of Defense Technology
Email:{rongchunli, zhouxin17a, hengyuepan, shyou, yongdou}@nudt.edu.cn

***Abstract*—In this paper, we propose a GPU-based QC-LDPC decoder for 5G New Radio(NR). Different from existing LDPC decoders based on GPUs, our decoder achieves high throughput when decoding LDPC codes with high code rates. Moreover, we implement the shortening and puncturing techniques which are exploited by 5G NR. The decoding algorithm Min-Sum approximation algorithm(MSA) is optimized to implement efficient parallel decoding on the GPU. In order to save the on-chip and the off-chip bandwidth, we propose the two-level quantization scheme and implement data packing on the GPU. We also analyse the optimum thread assignment for different code rates based on our implementation. By using the optimum settings on the GPU, the decoding throughput achieves 1.38 Gbps in the case of (2080, 1760), r=5/6 on Nvidia RTX 2080Ti.**

***Index Terms*—LDPC, 5G New Radio, GPU, SDR**

I. Introduction

Low-density parity-check(LDPC) codes [1] are a class of forward error correction(FEC) codes. LDPC codes have already been adopted by current communications systems such as WiFi, DVB-S2, and WiMAX because of their near-optimum performance. LDPC codes are also included in 5G standards and become the channel coding scheme of fifth-generation New Radio(5G NR). Although many high-throughput LDPC decoders have been proposed and perform well [2]–[6], new situations emerge when decoding LDPC codes for 5G NR. Firstly, as described in the standards delivered by the 3rd Generation Partnership Project(3GPP), the speed of the 5G wireless system will achieve 20 Gbit/s [7]. Throughput of encoding and decoding must be high enough to meet the demand of high-speed data transmission. It is necessary for LDPC decoders to improve decoding throughput. Secondly, the new standards require supports for a wider range of code rates and code lengths, and some new structure features such as the code design and decoding techniques are exploited to achieve better performance. Therefore, high-throughput LDPC decoders for 5G NR are needed when applying 5G NR.

Decoding throughput based on dedicated hardware platforms such as ASIC and FPGA can be very high [8]. However, it is known that GPU-based LDPC decoders can also achieve high throughput and even outperform FPGAs in some cases [9]. Because GPUs have strong computing capability, GPU-based LDPC decoders have great potential. One of the advantages of GPU decoders is its good flexibility. Multi-standard and multi-mode communication systems can be implemented by software using GPU-based decoders. Supports of different code rates and code lengths of LDPC decoding can be easily implemented by GPU-based decoders. Furthermore, soft defined radio(SDR) platforms are playing important roles in modern communications systems. Decoding by GPUs is an ideal solution to build SDR systems. In addition, short development time also makes GPU-based LDPC decoders very attractive.

WiFi and WiMAX systems exploit quasi-cyclic LDPC(QC-LDPC) codes which are structured LDPC codes with low complexity of encoding and decoding. New QC-LDPC codes have been designed in 5G NR standards. Proposed GPU-based QC-LDPC decoders are implementations of specific code lengths such as (2304, 1152) for IEEE 802.16e [10] and (1944, 972) for IEEE 802.11n [4], [5]. However, there are few implementations of high-throughput QC-LDPC decoders that support longer code lengths and higher code rates required by 5G NR.

In this paper, we propose a high-throughput GPU-based QC-LDPC decoder for 5G NR. In order to achieve high decoding throughput, we implement GPU-adapted parallel MSA. The minimum value and sign computations during message updating processes are optimized for parallel decoding on GPUs. In addition, we implement quantization of log-likelihood ratios(LLRs) and intermediate values to save on-chip and off-chip bandwidth with a reasonable performance penalty. We also exploit data-packing methods to reduce the overhead of data transmission between the GPU and the CPU. Throughput of different code lengths and code rates are measured on Nvidia RTX 2080 Ti and our GPU-based decoder performs well in most cases. Finally, we propose an evaluation method to obtain appropriate settings of the parallel decoding on the GPU.

The remainder of this paper is organized as follows. Section II introduces the construction of QC-LDPC codes for 5G NR. Section III describes the min-sum decoding algorithm. A GPU implementation of the LDPC decoder is proposed in Section IV and the experiment results are showed in Section V. Finally, Section VI gives the conclusions.

[1]These authors contributed equally to this work.

978-1-7281-8086-1/20/$31.00 ©2020 IEEE

TABLE I
LIFTING SIZES FOR 5G NR

a	Set of lifting sizes(I_L)	Range of j
2	{2, 4, 8, 16, 32, 64, 128, 256}	j = 0, 1, 2, 3, 4, 5, 6, 7
3	{3, 6, 12, 24, 48, 96, 192, 384}	j = 0, 1, 2, 3, 4, 5, 6, 7
5	{5, 10, 20, 40, 80, 160, 320}	j = 0, 1, 2, 3, 4, 5, 6
7	{7, 14, 28, 56, 112, 224}	j = 0, 1, 2, 3, 4, 5
9	{9, 18, 36, 72, 144, 288}	j = 0, 1, 2, 3, 4, 5
11	{11, 22, 44, 88, 176, 352}	j = 0, 1, 2, 3, 4
13	{13, 26, 52, 104, 208}	j = 0, 1, 2, 3, 4
15	{15, 30, 60, 120, 240}	j = 0, 1, 2, 3, 4

II. QC-LDPC CODES FOR 5G NR

A. Parity-check Matrix of QC-LDPC Codes

LDPC codes are constructed by a $M \times N$ parity-check matrix $\mathbf{H}$, where M represents the number of check nodes(CNs), and N represents the number of variable nodes(VNs). Denote K as the length of info bits and $K = N - M$. Each row in the parity-check matrix(PCM) represents a parity check equation as follows,

$$c_1 \oplus c_2 \oplus ... \oplus c_k = 0 \tag{1}$$

where $\oplus$ means modulo 2 addition. $c_1, c_2, ..., c_k$ are non-zero bits in the row. The relationship between CNs and VNs can also be described in a Tanner graph [11]. The edge in the graph between CN_i and VN_j corresponds to a non-zero entry in the PCM which means $H(i,j) = 1$, where $i = 1, 2, ...M, j = 1, 2, ..., N$.

The PCM of QC-LDPC codes can be constructed according to a base matrix $\mathbf{H}_b$. $\mathbf{H}_b$ is a $m_b \times n_b$ matrix that composed of $Z \times Z$ right-shifted identity and zero matrices. The 3GPP has defined two rate-compatible base matrices for NR, BG1 and BG2. BG1 is targeted for larger code lengths($500 \leq K \leq 8448$) and higher rates($1/3 \leq r \leq 8/9$), and BG2 is targeted for smaller code lengths($40 \leq K \leq 2560$) and lower rates($1/5 \leq r \leq 2/3$). Z is the lifting size which represents the size of right-shifted identity matrix. 3GPP uses 8 sets of lifting sizes for the basic graph. Value of $Z = 2^j \times a$, where $a \in \{2, 3, 5, 7, 9, 11, 13, 15\}$, and $j = 0, 1, 2, 3, 4, 5, 6, 7$. Table I shows the lifting sizes and the maximum Z is 384. Denote the shift value

$$s = \mathbf{H}_b(i_b, j_b) \in \{-1\} \cup \{0, 1, ..., Z-1\} \tag{2}$$

where $1 \leq i_b \leq m_b$ and $1 \leq j_b \leq n_b$. The PCM is constructed by expanding H_b using the mapping,

$$s \to \begin{cases} \mathbf{0}, & s = -1 \\ \mathbf{I}_s, & else \end{cases} \tag{3}$$

where $\mathbf{I}_s$ is an identity $Z \times Z$ matrix which is cyclically right-shifted by s. And $\mathbf{0}$ is the $Z \times Z$ zero matrix. The size of PCM $\mathbf{H}$ is $(m_b \times Z) \times (n_b \times Z)$. For BG1, $m_b = 46, n_b = 68$, and information columns $k_b = n_b - m_b = 22$. In 5G NR, m_b and n_b are adjustable to fit the code rate. Given a code rate r and a code length N to be encoded, the following steps describe the construction process of a PCM for QC-LDPC:

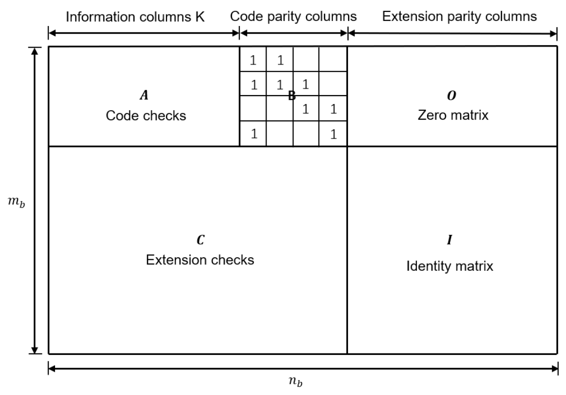

Fig. 1. Sketch of the base graph for 5G NR LDPC code

Step 1 Find the shortening length N_s which satisfies

$$\frac{k_b}{n_b - N_s - 2} - d < r < \frac{k_b}{n_b - N_s - 2} + d \tag{4}$$

The target code rate may not be satisfied exactly according to the construction method described above. We denote d as the maximum code rate deviation, the final code rate is allowed to have a slight deviation to the target code rate.

Step 2 Shortening base graph by eliminating the right-most N_s columns [12]. Then the size of shortened base graph is $(m_b - N_s) \times (n_b - N_s)$.

Step 3 Determine Z by selecting the minimum Z value in Table I, such that $k_b \times Z \geq K$. For BG1, $k_b = 22$.

Step 4 Construct the PCM by the right-shifted identity matrices and zero matrices according to the modified base graph as (3) describes.

The sketch of the base graph for 5G LDPC code is shown in Figure 1. In 5G QC-LDPC codes, shortening is carried out to obtain the desired information blocklengths and high code rates. The eliminated columns are extension parity-check parts.

B. Encoding of QC-LDPC for 5G NR

Given a certain PCM $\mathbf{H}$, the objective of LDPC encoding is to solve the parity equations:

$$\mathbf{H}\mathbf{c}^{\mathbf{T}} = \mathbf{0} \tag{5}$$

where $\mathbf{c}$ is the encoded codeword, which composed of the information bits and parity check bits. Encoding methods of QC-LDPC include Gaussian elimination, RU method proposed by Richardson and Urbanke [13], and efficient QC-LDPC encoder designed for 5G NR proposed in [14]. Once the PCM is determined, the generator matrix $\mathbf{G}$ can be derived from $\mathbf{H}$. Then the encoded codewords can be generated by

$$\mathbf{c} = \mathbf{u}\mathbf{G} \tag{6}$$

where $\mathbf{u}$ is the information bit array to be encoded.

Notice that the encoded LDPC codewords are punctured before transmission in 5G NR. The first $2Z$ bits are always punctured and not transmitted [15].

III. Decoding Algorithm of QC-LDPC

LDPC codes can be decoded using belief propagation(BP) method. The basic decoding algorithm of LDPC is sum-product algorithm(SPA) [16]. The approximate algorithm of SPA is scaled min-sum algorithm(MSA). Because the MSA has lower complexity than SPA [17], it is more suitable for pratical implementation. Denote y_j as received value and v_j as the j^{th} bit of the code. For $1 \le i \le M$ and $1 \le j \le N$, let the variable-to-check(VTC) message from VN_j to CN_i be L_{ij} and let the check-to-variable(CTV) message from CN_i to VN_j be R_{ij}. The decoding process iterates from 1^{st} to M^{th} layer. R and L are updated according to Z rows of elements in each layer. Denote l as the layer index of the message matrix and $l = 1, ..., m_b$. Becasue the $\mathbf{I}_s$ is an identity matrix, there is only one connection between VN_j and other CNs within each layer, the L_{ij}^l is replaced by L_j^l to simplify the computations. The initial value of L_j is a posteriori probability(APP) ratio for VN_j. The steps of decoding are described as follows:

1) Initialize the APP ratio and the CTV messages,

$$L_j^0 = \ln \frac{P\{v_j = 0|y_j\}}{P\{v_j = 1|y_j\}}, 1 \le j \le N, \tag{7}$$

$$R_{ij} = 0, 1 \le i \le M, 1 \le j \le N \tag{8}$$

2) Calculate L_j^l and R_{ij} by layers. Fisrtly, update L_j^l of all VNs from i^{th} using

$$L_j^{old,l} = L_j^{l-1} - R_{ij}^{old,l} \tag{9}$$

Secondly, Update R_{ij}^l by

$$R_{ij}^{new,l} = \alpha \prod_{k \in \mathcal{N}(i)\backslash\{j\}} sign(L_k^l) \min_{k \in \mathcal{N}(i)\backslash\{j\}} \{|L_k^l|\} \tag{10}$$

where, $1 \le i \le M$, $k \in \mathcal{N}(i)\backslash\{j\}$ represents the set of the VN neighbors of CN_i excluding VN_j and α is the scale factor. Thirdly, Update L_j^l by

$$L_j^{new,l} = L_j^{old,l} + R_{ij}^{new,l} \tag{11}$$

3) After calculated all layers of CNs and all the VNs for $1 \le l \le m_b$ and $1 \le j \le N$, the information bits are determined by results of the last layer where $l = m_b$,

$$v_j = \begin{cases} 0, & L_j^{m_b} \ge 0 \\ 1, & else \end{cases} \tag{12}$$

IV. High-throughput GPU-based LDPC Decoder for 5G NR

In this section, we describe several techniques for high-throughput decoding for 5G NR. We focus on accelerating the decoding process of QC-LDPC codes which are constructed using BG1. These codes are featuring high code rates and large code lengths. The decoding algorithm MSA is a layered decoding algorithm and can be paralleled to decode efficiently on GPUs. In order to achieve high-throughput decoding, it is necessary to further optimize the decoding process. Moreover, high code rates and large code lengths have challenged the memory bandwidth on GPUs. In general, the overhead of memory access affects the decoding speed. The size of the parity-check matrix increases significantly in 5G NR. As a result, the demand for memory resources on GPU increases correspondingly. Therefore, a reasonable assignment of resources on GPUs is essential when decoding.

A. Compressed Data Structures

The parity-check matrix $\mathbf{H}$ is a sparse matrix. Generally, we exploit a compressed structure instead of transmitting the whole PCM to the GPU. The common method to locate non-zero entries in $\mathbf{H}$ is generating a look-up table before decoding. We can store positions of non-zero entries in each row. However, the number of non-zero entries in each row of $\mathbf{H}$ is not identical. It is a wasting of time to execute the same number of iterations if there are few non-zero entries in most rows. Consequently, we not only record positions of non-zero bits in each row, but the number of non-zero entries is also recorded.

Denote P_{ik} as the index of k^{th} non-zero entry in i^{th} row in $\mathbf{H}$. Because there are at most k_b non-zero entries in each row in $\mathbf{H}$, then $1 \le k \le k_b$. Let C_i as the number of non-zero entries of i^{th} row, then the connection between CNs and VNs can be determined by P_{ik} and C_i. Compressing structure of $\mathbf{H}$ saves memory space and accessing time. Moreover, compressing is easy to implement on GPUs.

In addition to compressing of $\mathbf{H}$, the CTV messages $\mathbf{R}$ can also be compressed. As described above, connections between VNs and CNs are known before decoding. Therefore, we only need to save message values of VNs in i^{th} row if there are connections betweenVN_j and CN_i. Consequently, the size of $\mathbf{R}$ can be compressed from $(m_b \times Z) \times (n_b \times Z)$ to $(m_b \times Z) \times k_b$. Notice that compressing of $\mathbf{R}$ is also an optimization method of memory coalescing on the GPU. The CTV messages are intermediate results during decoding. Compressing $\mathbf{R}$ to continuous space helps reduce bank conflicts within a thread warp on GPUs.

B. Parallel Layered Decoding

In this study, we exploit MSA because it is suitable for parallel decoding on GPUs. As described above, the PCM is constructed using right-shifted identity matrices. For each $Z \times Z$ block in $\mathbf{H}$, all non-zero entries are from different columns. Therefore, there is no dependency when calculating CTV and VTC messages from $i \times Z$ to $(i+1) \times Z$ layer, where $i = 0, 1, ...m_b - 1$. Based on this, the thread assignment of decoding is determined. We allocate Z threads for the layered decoding of MSA. The maximum Z is 384 for 5G NR and it does not exceed the maximum number of threads per block on the GPU.

As stated above, R_{ij} and L_{ij} are updated by (10) and (11). The complexity of computing minimum value from $|L_{ik}^l|, k \in \mathcal{N}(i)\backslash\{j\}$ is $\mathcal{O}(d_i^2)$, where d_i is the number of non-zero elements of i^{th} row. A linear complexity method is proposed in [3]. A two-pass process is implemented in this method. The first pass is the global pass and the first and the second minimum value are calculated. In the local pass, the minimum value of $|L^l_{ik}|$ is determined according to values obtained in the global pass.

Denote f, s as the 1st and 2nd minimum value of $|L_{ij}|, j \in \mathcal{N}(i)$. Set initial values of f and s to positive maximum values. The first and second minimum value among all the VNs f^l and s^l are obtained in the first pass by

$$f^l = \begin{cases} |L_{ik}|, & |L_{ik}| < f^{l-1} \\ f^{l-1}, & else \end{cases} \tag{13}$$

and

$$s^l = \begin{cases} |L_{ik}|, & f^{l-1} < |L_{ik}| < s^{l-1} \\ f^{l-1}, & |L_{ik}| < f^{l-1} \\ s^{l-1}, & else \end{cases} \tag{14}$$

Then we can get the minimum value of VN_j by

$$L_{min} = \begin{cases} f^l, & |L_{ik}| \neq f^l \\ s^l, & else \end{cases} \tag{15}$$

Similarly, denote $\mathcal{S}_{global} = \prod_{k\in\mathcal{N}(i)} sign(L^l_{ik})$ and $\mathcal{S} = \prod_{k\in\mathcal{N}(i)\setminus\{j\}} sign(L^l_{ik})$. In the global pass, calculate $\mathcal{S}_{global}$ by

$$\mathcal{S}_{global} = \begin{cases} -sign, & L_{ik} < 0 \\ sign, & else \end{cases} \tag{16}$$

, then obtain $\mathbf{S}$ by

$$\mathcal{S} = \begin{cases} -\mathcal{S}_{global}, & L_{ik} < 0 \\ \mathcal{S}_{global}, & else \end{cases} \tag{17}$$

Because only negative values affect sign of $\mathcal{S}_{global}$, we only update the $\mathcal{S}_{global}$ when $L_{ik} < 0$. Meanwhile, $\mathcal{S} = \mathcal{S}_{global}$ if $L_{ik} > 0$. Based on this, the multiplication in (10) is replaced by (16) and (17). This replacement reduces computations on the GPU.

The inputs of Algorithm 1 are llr calculated by (7), the index matrix of non-zero entries P and number of non-zero entries of each layer C. L is the vectors of intermediate LLRs. f and s are vectors of 1st and 2nd minimum values. Because there is no dependency when processing messages of entries in the $(n_b \times Z) \times Z$ block, calculating of L are processed parallel by Z threads in our decoder. In BG1 of 5G NR, the shift values from different layers may be the same in some cases. Therefore, more threads of allocation for a single code is not suggested. Consequently, the number of iterations of the layer is m_b. We found that the minimum value is 0 for all the VNs in the first iteration. Therefore, the start layer is $l = 1$ in our implementation to reduce decoding time.

In the parallel decoder, we implement the two-phase method to simplify the process of finding minimum values and the product of signs. This method eliminates the nested loops and improves the decoding throughput significantly.

Algorithm 1 Parallel Layered Decoding

```
Require: $llr$, $P$, $C$
Ensure: $v$
1: Initialization, $R \leftarrow 0$
2: $L[2Z \dots m_b \times Z-1] \leftarrow llr[0 \dots (Nb-2) \times Z]$ //Puncturing
3: for $iter = 0 \to ITER$ do
4:    for layer $l = 1 \to m_b$ do
5:       for thread $t = 0 \to Z-1$ parallel do
6:          $f^{l,t} \leftarrow$ MAX, $s^{l,t} \leftarrow$MAX
7:          //Global pass
8:          for $j = 0 \to C^{l,t}$ do
9:             $k \leftarrow P(l,j)$
10:            $L(t,k) \leftarrow L(t,k) - R^l(t,k)$
11:            Calculate $f^l(t), s^l(t)$
12:            Update $\mathcal{S}_{global}(t)$
13:         end for
14:         //Local pass
15:         for $j = 0 \to C^{l,t}$ do
16:            Calculate $|L_{min}|(t)$ using $f^l(t), s^l(t)$
17:            Update $\mathcal{S}(t)$
18:            $k \leftarrow P(l,j)$
19:            $L(t,k) \leftarrow L(t,k) + \alpha \times |L_{min}(t)| \times \mathcal{S}(t)$
20:         end for
21:      end for
22:   end for
23: end for
24: //Hard Decisions are executed parallel by Z threads.
```

C. Multi-chunk Decoding

There are Z threads decoding in parallel per block as staged above. However, the maximum number of threads per block is 1024 or 2048 on the GPU. Although the number of threads exceeding Z is not suggested, the maximum Z is 384 which is far from exceeding the maximum limit of threads per block. Therefore, it is possible that more threads can be allocated to decode multiple codes within the same block. Therefore, we implement multi-chunk decoding to achieve more parallelism on the GPU. Let N_p as the number of codes being decoded in the same block. Then the number of threads per block is $N_p \times Z$. Consequently, denote N_c as the number of input codes, we allocate N_c/N_p blocks for the decoding kernel. There are N_c/N_p chunks being decoded together and each chunk is decoded by $N_p \times Z$ threads.

D. Memory Hierarchy

In order to make the best use of memory resources on the GPU, frequently accessed values are stored in shared memory. In our implementation, f, s, and L are stored into shared memory. The size of them are $1 \times Z$, $1 \times Z$ and $Z \times n_b$ respectively. It is not suggested to put R into shared memory. Because the size of R is $(m_b \times Z) \times k_b$. $k_b = 22$ in BG1 and the maximum $m_b = 46$. Size of R is so large that putting it in shared memory may limit the number of resident blocks when decoding. Instead, R is stored in global memory.

Notice that the size of L is only $n_b \times Z$. We only care about the final results so intermediate results are overwritten in the latter iteration of layers. L is frequently accessed when decoding. Moreover, the largest $n_b = 68$ and $Z_{max} = 384$ in 5G NR. The size of L is acceptable. Therefore, we store values of L into shared memory in this implementation.

E. Two-level Quantization

Many quantization schemes have been proposed to improve the decoding throughput of LDPC [17]. In our implementation, in order to utilize the bandwidth more efficiently with less penalty of performance, we propose a two-level quantization scheme.

In general, the smallest length of built-in types supported by GPUs is 8-bit [18]. Quantization schemes less than 8-bit are not suitable for high-throughput decoding based on GPUs. Because less length may introduce many type conversions that are executed to fit the minimum length supported by native arithmetic instructions in the CUDA. Type conversions have low throughput on the GPU compared to other arithmetic operations such as addition and multiplication. Moreover, the bitwise operations like AND, XOR as well as shifting also have low throughput [18].

Therefore, we use 8-bit values during the decoding process to save bandwidth. Although computations of data types less than 8-bit are not efficient, the efficiency of transmission between the CPU and GPU can be improved by data packing. Based on this, we quantify the llr to 4-bit number at the first level. And L and R are quantified to 8-bit number at the second level. The first bit is the sign bit on both levels.

There are four advantages of our two-level quantization scheme.

1) The first level quantization, 4-bit llr scheme further saves the off-chip bandwidth. Communication latency between the CPU and GPU can be reduced.
2) The 4-bit scheme avoids overflows during the computing of L. As described in (11), L is added with R in each iteration. It is possible that the number of L overflows after several iterations. And our experimental results have also proved that the overflows are common when decoding with an 8-bit data type. Therefore, we quantify llr to 4-bit to limit the range of initial values. This scheme avoids most overflows during the computation.
3) The second level quantization of L and R saves on-chip bandwidth. In order to achieve high-throughput decoding, on-chip bandwidth is also an important factor to be considered.
4) The second level quantization also saves space of shared memory and registers. The resident blocks are limited by the occupancy of shared memory. Saving shared memory and registers helps achieve more parallelism.

The penalty for performance of the two-level scheme can be ignored compared to the 8-bit scheme.

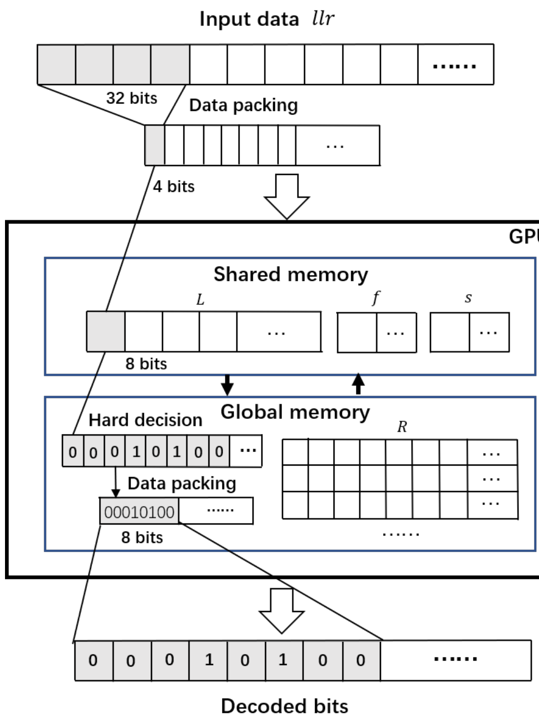

Fig. 2. Memory hierarchy and data packing

F. Data packing

Data packing can reduce decoding latency caused by data transmission between the CPU and GPU. The outputs of the decoder is binary code bits. Although it is inefficient to implement bitwise operations on GPUs, it is easy to find that we can pack the decoding results to save bandwidth. In our implementation, every 8 decoded bits are packed to an 8-bit char value after the hard decision. Then we transmit the packed data to the CPU.

In addition to the packing of decoded bits, we combine quantization and data packing to further save the bandwidth. As described above, the input llr values are quantified to 4-bit fix numbers. Therefore, every two values of llr can be packed to an 8-bit char value. Once the packed data are transmitted to the GPU, they are converted to 8-bit char values for the next decoding process.

Figure 2 shows the memory hierarchy and data packing of the proposed decoder. The input data are quantified to 4-bit values before decoding. The intermediate values are 8-bit values. L, f and s are stored in shared memory. R is stored in global memory. 8 decoded information bits are packed into one char type value. Then the packed data are transmitted to the CPU.

V. Experimental Results

A. Experimental setup

In this paper, the QC-LDPC decoder is implemented on Turing architecture based NVIDIA RTX 2080Ti which has 4352 CUDA cores, 68 multiprocessors and 11GB of GDDR6

TABLE II
BASE GRAPH OF DIFFERENT CODE RATE

$\mathbf{r}$	$(\mathbf{N},\mathbf{K})$	$\mathbf{m_b}\times\mathbf{n_b}$	$\mathbf{Z}$
$1/3$	$(2112, 704)$	46×68	32
$1/2$	$(2160, 1056)$	25×47	48
$2/3$	$(2112, 1408)$	13×35	64
$3/4$	$(2088, 1584)$	9×31	72
$5/6$	$(2080, 1760)$	6×28	80

TABLE III
THROUGHPUT OF $(2080, 1760), r = 5/6$, (ITERATION = 10)

Method	**Throughput(Mbps)**	**Speedup**
Floating	267	-
8-bit	557	2.1x
8-bit multi-chunk	661	3.1x
8-bit multi-chunk & data packing	773	3.3x

memory. The CPU is Intel i7-8700K. The CUDA version is 10.2.

The Quadrature Phase Shift Keying(QPSK) modulation is exploited when generating encoded values. The additive white Gaussian noise(AWGN) is added to the encoded values. The scale factor of MSA $\alpha = 0.75$ and $E_b/N_0 = 3dB$. The maximum code rate deviation $d = 0.15$. Early termination(ET) is not applied when measuring the decoding throughput so the parameters above are set to common values.

B. Decoding throughput

As stated above, we first provide the code length and code rate, then the appropriate lifting size is selected from Table I. Correspondingly, the base graph is shortened to fit the code rate. As a result, the base graph varies according to the code rate. Table II describes the structure of the base graph for different code rates. In this paper, we aim at improving the decoding throughput of QC-LDPC codes constructed by BG1. And the base code length is 2048. As shown in the table, the lifting size increases with code rates. The m_b and n_b decreases because of the shortening of H_b.

The throughput T is defined as $T = \frac{N_c}{t}$, where N_c is the number of code blocks and t is the decoding time.

We measure the throughput of the proposed decoder which are implemented using different optimizing methods. As the results shown in Table III, the decoding throughput increases massively when implementing the proposed two-level quantization method. Moreover, the data packing scheme also has a good effect. The improvement of throughput is more than 15% compared to the method without data packing.

Decoding throughputs of different code rates in Table II are also measured. The parity-check matrix $\mathbf{H}$ is shortened for high code rates. As shown in Table II, the sizes of m_b and n_b decrease with increasing of the code rate. The number of iterations decreases correspondingly both in the horizontal and vertical directions. As a result, the decoding time of our parallel decoder decreases, and the throughput increases significantly. Figure 3 describes the decoding throughput of different code rates. The throughput of multi-chunk decoding

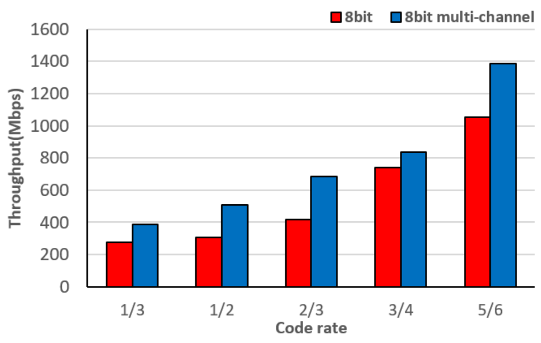

Fig. 3. Throughput of different code rates

TABLE IV
OCCUPANCY OF RESOURCES ON THE GPU

Name	**value**
X	$\min(\frac{R_{SM}}{N_r}, \frac{M_{SM}}{N_s}, \frac{T_{SM}}{N_t}, B_{SM})$
B_{SM}	16
N_{SM}	68
N_r	63
N_t	Z
N_s	$(N_b + 2)\times Z$

is much larger than the throughput of normal decoding. The average improvement is more than 40%.

In order to make the most efficient use of resources on the GPU, the assignment of threads is adjusted according to the code rate. It is well known that the number of registers and shared memory for each thread block and each multiprocessor is limited. In addition, the maximum of resident blocks and threads per SM is also considered to avoid overloading. Denote N_p, N_r, N_s, N_t as the number of codes, registers, shared memory, and threads per block. And denote $R_{SM}, M_{SM}, T_{SM}, B_{SM}$, as maximum registers, shared memory, resident threads and resident blocks per SM. N_{SM} is the number of SMs on the GPU. Let N_b as resident blocks on each SM. Then the appropriate thread assignment is obtained as follows:

$$N_b N_p \leq \min(\frac{R_{SM}}{N_r}, \frac{M_{SM}}{N_s}, \frac{T_{SM}}{N_t}, B_{SM}) \tag{18}$$

and the number of code blocks being decoded simultaneously can be evaluated by

$$N_c \leq N_b N_p N_{SM} \tag{19}$$

Based on this, we evaluate the most appropriate thread assignment and the number of code blocks being decoded parallel. Let $X = \min(\frac{R_{SM}}{N_r}, \frac{M_{SM}}{N_s}, \frac{T_{SM}}{N_t}, B_{SM})$ and $N_c^t = N_b N_p N_{SM}$. For RTX 2080 Ti, the settings in our implementation are described in Table IV. N_r is obtained by the $nvcc$ compiler. Because the length of input data llr, $L_i = N_b \times Z$ and output data $L_o = Z \times k_b$ vary according to Z and N_b. The transmission time is not included when calculating the throughput. Table V describes the thread assignment implemented on RTX 2080 Ti. The optimum number of code

TABLE V
OPTIMUM THREAD ASSIGNMENT AND THROUGHPUT

r	**X**	$\mathbf{N}_c^t$	$\mathbf{N}_c$	$\mathbf{N}_p$	**Throughput(Mbps)**
1/3	16	1088	1000	5	389
1/2	16	1088	1000	5	506
2/3	16	1088	1000	5	684
3/4	14	952	1000	5	838
5/6	12	816	800	2	1385

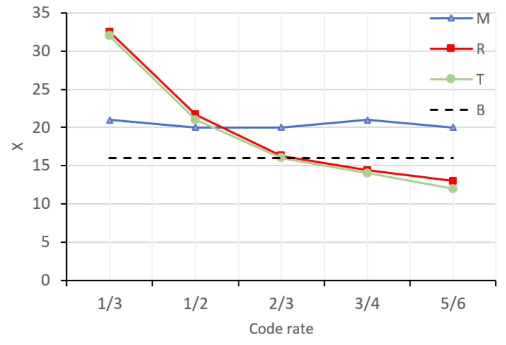

Fig. 4. Maximum resident blocks per SM

blocks being decoded together N_c^t is obtained by (18) and (19). And we select an appropriate N_p to achieve a higher throughput.

Fig 4 shows how the maximum resident blocks per SM X is affected by the code rate on the GPU. M, R, T, B represent shared memory, registers, resident threads, and resident blocks per SM respectively. The lifting size Z is small in low code rate cases. The main restriction is the maximum resident blocks per SM. As for high code rates cases, Z is larger and N_p and N_c are restricted by the maximum amount thread per block, and the maximum amount of shared memory.

C. Comparison with previous works

There are few existed GPU-based QC-LDPC decoders for 5G NR. The number of information columns k_b is larger in 5G NR. For example, the maximum non-zero entries in WiMax are 7 while the number is 22 in 5G NR. Although more iterations are processed when decoding, our proposed decoder achieves Gbits per second throughput for high code rates. In Table VI, the number of iterations is 10 and the decoding algorithm is layered decoding of MSA, the throughput in the table are represented by Mbps.

VI. CONCLUSIONS

In this paper, a GPU-based LDPC decoder for 5G NR is proposed. We implement the parallel MSA on the GPU. The decoding process is optimized to make the most efficient use of resources on the GPU. We exploit the multi-chunk decoding method to implement the parallel decoding within a code block. A two-level quantization scheme is proposed to further save the on-chip and off-chip bandwidth on the GPU. The high throughput is achieved by evaluating the optimum thread assignment before decoding. Our decoder achieves a throughput of 1.38 Gbps in the case of $(2080, 1760), r = 5/6$.

TABLE VI
COMPARISON WITH PROPOSED WORKS

Work	**Standard**	$(\mathbf{N}, \mathbf{K})$	**r**	**GPU**	**Throughput**
[2]	WiMax	$(2304, 1152)$	1/2	GTX 580	620
[10]	WiMax	$(2304, 1152)$	1/2	GTX Titan	316.07
[4]	WiFi	$(1944, 972)$	1/2	Titan X	913
[5]	WiFi	$(1944, 972)$	1/2	GTX 1080Ti	971.5
Ours	5G NR	$(2112, 1408)$	2/3	RTX 2080Ti	684
		$(2080, 1760)$	5/6		1385

ACKNOWLEDGEMENT

This paper is supported by the National Key Research and Development Program of China (Grant No. 2018YFB1003405) and the National Natural Science Foundation of China (Grant No. 61802419).

REFERENCES

[1] R. Gallager, “Low-density parity-check codes,” *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.
[2] R. Li, Y. Dou, D. Zou, S. Wang, and Y. Zhang, “Efficient graphics processing unit based layered decoders for quasicyclic low-density parity-check codes,” *Concurrency and Computation: Practice and Experience*, vol. 27, no. 1, pp. 29–46, 2015.
[3] S. Mhaske, H. Kee, T. Ly, A. Aziz, and P. Spasojevic, “High-throughput fpga-based qc-ldpc decoder architecture,” in *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*. IEEE, 2015, pp. 1–5.
[4] S. Keskin and T. Kocak, “Gpu-based gigabit ldpc decoder,” *IEEE Communications Letters*, vol. 21, no. 8, pp. 1703–1706, 2017.
[5] J. Yuan and J. Sha, “4.7-gb/s ldpc decoder on gpu,” *IEEE Communications Letters*, vol. 22, no. 3, pp. 478–481, 2017.
[6] S. Guo, Y. Dou, Y. Lei, R. Li, and Y. Li, “An efficient multi-standard qc-ldpc decoder based on the row-layered decoding algorithm,” *IEICE Electronics Express*, pp. 12–20 150 356, 2015.
[7] 3GPP, “Release 15 for 5g new radio,” https://www.3gpp.org/release-15.
[8] Y. Wang, Q. Wang, Y. Zhang, S. Qiu, and Z. Xing, “An area-efficient hybrid polar decoder with pipelined architecture,” *IEEE Access*, vol. 8, pp. 68 068–68 082, 2020.
[9] G. Falcão, V. Silva, and L. Sousa, “How gpus can outperform asics for fast ldpc decoding,” in *Proceedings of the 23rd international conference on Supercomputing*, 2009, pp. 390–399.
[10] G. Wang, M. Wu, B. Yin, and J. R. Cavallaro, “High throughput low latency ldpc decoding on gpu for sdr systems,” in *2013 IEEE Global Conference on Signal and Information Processing*. IEEE, 2013, pp. 1258–1261.
[11] R. Tanner, “A recursive approach to low complexity codes,” *IEEE Transactions on information theory*, vol. 27, no. 5, pp. 533–547, 1981.
[12] T. Richardson and S. Kudekar, “Design of low-density parity check codes for 5g new radio,” *IEEE Communications Magazine*, vol. 56, no. 3, pp. 28–34, 2018.
[13] T. J. Richardson and R. L. Urbanke, “Efficient encoding of low-density parity-check codes,” *IEEE transactions on information theory*, vol. 47, no. 2, pp. 638–656, 2001.
[14] T. T. B. Nguyen, T. Nguyen Tan, and H. Lee, “Efficient qc-ldpc encoder for 5g new radio,” *Electronics*, vol. 8, no. 6, p. 668, 2019.
[15] J. H. Bae, A. Abotabl, H.-P. Lin, K.-B. Song, and J. Lee, “An overview of channel coding for 5g nr cellular communications,” *APSIPA Transactions on Signal and Information Processing*, vol. 8, 2019.
[16] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Transactions on information theory*, vol. 47, no. 2, pp. 498–519, 2001.
[17] J. Chen, A. Dholakia, E. Eleftheriou, M. P. Fossorier, and X.-Y. Hu, “Reduced-complexity decoding of ldpc codes,” *IEEE transactions on communications*, vol. 53, no. 8, pp. 1288–1299, 2005.
[18] C. Nvidia, “Nvidia cuda c programming guide,” *Nvidia Corporation*, vol. 120, no. 18, p. 8, 2011.