

A High Throughput and Flexible Rate 5G NR LDPC Encoder on a Single GPU

Shixin Liao^{*†}, Yueying Zhan^{*}, Ziyuan Shi^{*†} and Lei Yang^{*}

^{*}Key Laboratory of Space Utilization, Technology and Engineering Center for Space Utilization,
Chinese Academy of Sciences, Beijing, China

[†]University of Chinese Academy of Sciences, Beijing, China

liaoshixin19@csu.ac.cn, zhanyueying@csu.ac.cn, shiziyuan18@csu.ac.cn, yang.lei@csu.ac.cn

Abstract—In order to build a high performance low-density parity-check (LDPC) communication link simulation platform, high speed LDPC encoding for information sequence is required. In this paper, a high and flexible throughput LDPC encoding implementation based on a single GPU is proposed. We discuss the parallelism of the LDPC encoding algorithm employs the core parity check bits and single diagonal parity check bits for the fifth generation new ratio. We implement the parallel LDPC encoder on CUDA platform. The experimental results show that our LDPC encoding module achieves a 38–62Gbps throughput for the rate from 1/2 to 8/9 on a single GPU. The results also demonstrate that parallel simulation tasks based on GPUs can achieve a good trade-off between performance and cost.

Index Terms—5G New Ratio, quasi-cyclic LDPC code, channel encoding, parallel computing, CUDA

I. INTRODUCTION

Low-density parity-check (LDPC) codes are first proposed by Gallager [1]. As LDPC codes technology is the most promising channel coding to achieve the error correction capabilities close to Shannon limit, it was included in the key protocol of the fifth generation (5G) wireless communication standard [2]. However, there are still many problems about LDPC code that have not been solved thoroughly. To name but one example, how to design high performance LDPC matrices is the core problem in the field of channel coding, which determines the performance of LDPC code in the actual communication system. The search space of designing LDPC matrices is huge and the block error rate of LDPC code down to 10^{-7} . To evaluate the performance of LDPC codes on a link simulation platform in a reasonably short period, highly parallel LDPC codes simulation platforms have to be explored.

LDPC encoding involves complex matrix operations, and it has great influence on the efficiency of simulation. The most general method is to derive the generator matrix G from parity check matrix H by Gaussian elimination. However, this method contains a complexity $O(N^2)$ so it is too slow to be used in high performance applications. In the past decade, researchers and engineers try to accelerate LDPC encoding by improving encoding algorithm and using high performance computing platform.

When it comes to high performance computing, the hardware platforms we usually use are FPGAs, ASICs and GPUs. After T. Richardson and R. Urbanke proposed the RU encoding algorithm [3], many researchers have implemented the

RU encoding algorithm and its different improved versions on FPGAs [4]–[6]. The highest throughput among them achieves 10Gbps with the clock frequency of 125 MHz. The drawback of RU algorithm is that its column permutations may cause non-systematic codewords, thus reducing the efficiency of receiver decoding. Others implement the proposed algorithm on FPGAs. A. Mahdi and V. Paliouras implement encoding algorithm based on LU decomposition on a FPGA and they achieve 850Mbps throughput [7]. A. A. A. Hariri et al. implement a LDPC encoder uses a calculation window method which achieves up to 29Gbps data rates on a FPGA [8]. However, the further improvement of the performance of their algorithms on FPGAs requires vast XOR gates and shift registers, which is expensive.

In [9], T. T. B. Nguyen proposed a parallel LDPC encoding algorithm for 5G new ratio (5G NR). He implemented this method on ASICs and the throughput ranged from 22.1–202.4 Gbps. His algorithm has linear complexity and high parallelism. However, ASICs require expensive cost for taping out. Developers must produce ASIC chips for each submatrix size and each code rate to achieve the best performance, while software can provide a flexible submatrix size, with better maintainability and expansibility.

Compared with FPGAs and ASICs, GPUs have shorter development cycle and lower cost. Moreover, GPU-based applications can be flexibly modified and adjusted at a little expense of performance power ratio. Compute Unified Device Architecture (CUDA), which is a parallel computing platform and application programming interface (API) created by Nvidia, provides a way to exploit GPUs computation power in a C programming environment [10]. On CUDA, D. Kun implements a DVB-S2 LDPC encoder and decoder, which achieves 3–16Gbps for the code rate from 1/4 to 8/9 [11].

In this paper, we focus on high-speed 5G NR LDPC encoding. We build a 5G LDPC encoder on CUDA platform based on the algorithm T. T. B. Nguyen proposed. We analyze the parallelism of the encoding algorithm and discuss the best implementation. On a single GPU, our implementation achieves the information rate of 42Gbps for the code rate 1/3 and 62Gbps for the code rate 8/9 with $K = 1936$. Experimental results demonstrate that our algorithm has achieved the maximum parallelism allowed by the structure of 5G LDPC codes.

The remaining parts of this paper is organized as follows. Section II reviews the LDPC encoding process. Section III overviews memory features of the CUDA architecture GPU and the software programming model. Section IV introduces the encoding algorithm proposed to be implemented. In Section V, we analyze the parallelism of the algorithm and describe the implementation. The experimental results are presented in Section VI. Conclusions are drawn in Section VII.

II. LDPC ENCODING PROCESS

In a digital communication system, at the transmitter side the following procedure happens as Fig. 1. First, a large message block is divided into several code blocks with length K , i.e., message bits s . If the last block is less than K bits, pad zeros at the end. Second, encode each code block to a codeword C . Finally, transport codeword bits without the puncturing bits.

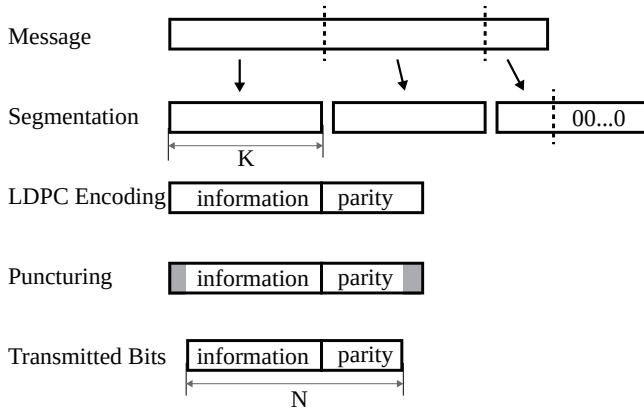


Fig. 1. 5G NR LDPC Encoder

The parity check matrices of LDPC codes determines the performance. Structured LDPC usually have very good performance of encoding and decoding. A lot of structured LDPC codes have been presented, such as the LDPC codes based on Euclidean geometry (EG), the LDPC codes based on balanced incomplete block design, protograph (PTG) LDPC codes, superposition (SP) LDPC codes and so on. In recent years, research on structured LDPC codes has been focused on quasi-cyclic (QC) LDPC codes. QC-LDPC codes allow the linear time complexity of encoding and decoding.

In QC-LDPC codes, the parity matrix H is composed of circulant permutation matrices. Let Z be the size of the circulant permutation matrix. Let α be the shift value of the $Z \times Z$ circulant permutation matrix. A $Z \times Z$ circulant permutation matrix $Q(\alpha)$ is a circulant matrix obtained by shifting the $Z \times Z$ identity matrix I to the right by α times over GF(2). For example,

$$Q(1) = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 1 & 0 & 0 & \dots & 0 \end{pmatrix} \quad (1)$$

In particular, $Q(-1)$ denotes the null matrix (all elements equal to zero) of the same size.

It is noteworthy that the computation of c satisfy a equations like $(A Q(\alpha))c^T = 0^T$ can be written as:

$$\begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ a_{2,1} & \dots & a_{2,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} + \begin{pmatrix} p_{\alpha+1} \\ \vdots \\ p_n \\ p_1 \\ p_2 \\ \vdots \\ p_\alpha \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad (2)$$

where A, α and s are known. It can be seen that we can work out a bit of p according to each equation independently. Therefore, these equations can be solved in parallel.

H is an $M \times N$ matrix, where $M = m_b \times Z$ and $N = n_b \times Z$. H is represented by the following $m_b \times n_b$ array of $Z \times Z$ circulants over GF(2):

$$H = \begin{pmatrix} Q(\alpha_{1,1}) & Q(\alpha_{1,2}) & \dots & Q(\alpha_{1,n_b}) \\ Q(\alpha_{2,1}) & Q(\alpha_{2,2}) & \dots & Q(\alpha_{2,n_b}) \\ \vdots & \vdots & \ddots & \vdots \\ Q(\alpha_{m_b,1}) & Q(\alpha_{m_b,2}) & \dots & Q(\alpha_{m_b,n_b}) \end{pmatrix} \quad (3)$$

III. GPU HARDWARE ARCHITECTURE AND CUDA PROGRAMMING MODEL

In CUDA architecture, the hardware resources on the GPUs determines the CUDA programming model. A CUDA architecture GPU has multiple cores that can run the same functions in parallel. When a function is executed on a core, it is called a kernel function. For programmers, kernel is multi-threaded and each thread can process different input parameters. This feature of CUDA is called thread-level parallelism. Therefore, the key to accelerating a program on the GPU is that the program should be highly parallel.

In addition, multiple cores are grouped together to create warps. All the cores in a warp are always running the same machine instructions. When a conditional branch is encountered, and the data in different cores satisfies different conditions, all the cores perform the operations of all the branches, and then selectively stores the data. Therefore, if there are complex conditional branches in kernel function, it will take more time than a single thread function.

CUDA provides many kinds of programmable memory, such as global memory, shared memory, and constant memory. Among these memory, global memory is the largest and slowest, and it can be read and written by the CPU and GPU. Shared memory is smaller and faster than global memory, can only be read and written by the GPU. Shared memory is often used to store intermediate variables. Constant memory is the fastest, but it can only be written by the CPU and read by the GPU. The relationship between cores and memories is shown in Fig. 2.

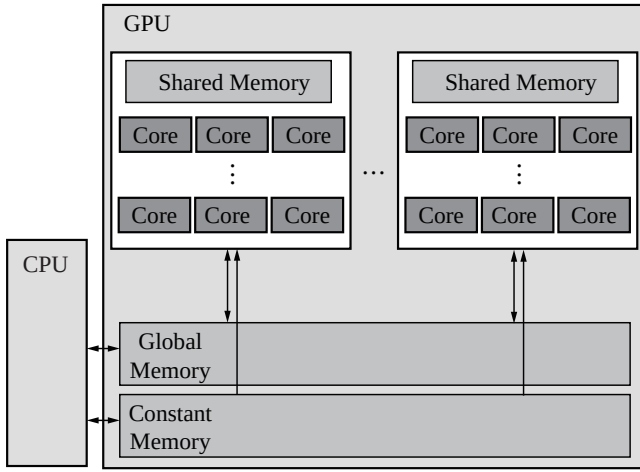


Fig. 2. Cores and memory resources in a GPU.

IV. 5G NR LDPC ENCODING ALGORITHM

The foundation for highly parallel computing on GPUs is that there are a large number of cores that can execute computation simultaneously. Therefore, only algorithms with high potential parallelism can be speeded up on GPUs. In this section, we describe the structure of 5G NR QC-LDPC parity matrices and introduce a 5G NR LDPC encoding algorithm.

A. 5G NR QC-LDPC Parity Matrices

QC-LDPC codes have been accepted as the channel coding scheme in 5G communications in 3GPP standard [2]. According to the 5G NR standard, there are two base graphs, BG1 and BG2, each base graph including 8 sets of parity matrices. Before starting encoding, we need to obtain a proper parity matrix H . In both BG1 and BG2, the parity matrix H is stored in the form of $m_b \times n_b$ shift coefficient matrix:

$$V(H) = \begin{pmatrix} V_{1,1} & V_{1,2} & \dots & V_{1,n_b} \\ V_{2,1} & V_{2,2} & \dots & V_{2,n_b} \\ \vdots & \vdots & \ddots & \vdots \\ V_{m_b,1} & V_{m_b,2} & \dots & V_{m_b,n_b} \end{pmatrix} \quad (4)$$

where $V_{i,j}$ is the (i,j) -th element correspond to the shift value $\alpha_{i,j}$ in (3). For a given Z , calculate $\alpha_{i,j}$ from $V_{i,j}$ as (5):

$$\alpha_{i,j} = \begin{cases} -1, & V = -1 \\ V_{i,j} \bmod Z, & \text{otherwise.} \end{cases} \quad (5)$$

Then replace $V_{i,j}$ by $Q(\alpha_{i,j})$ to obtain H .

Both of BG1 and BG2 consist of 8 sets of parity matrices correspond to different size Z . Z is the minimum integer that satisfies the following form:

$$\begin{aligned} Z &= a \times 2^j, \\ k_b \times Z &\geq K \end{aligned} \quad (6)$$

where a equals to 2, 3, 5, 7, 9, 11, 13 or 15. The set of shift coefficients are listed in Table I.

TABLE I
SETS OF 5G NR LDPC MATRICES

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

B. 5G NR LDPC Encoding Algorithm

Fig. 3 illustrates the general structure of the 5G NR QC-LDPC base graph.

The parity matrix H can be divided into 6 parts: A, B, O, C_1, C_2 and I . In different base graphs, A, C_1 and C_2 are different. O is a zero matrix and I is an identity matrix. B is a 4×4 matrix:

$$B = \begin{pmatrix} 1 & 0 & -1 & -1 \\ 0 & 0 & 0 & -1 \\ -1 & -1 & 1 & 0 \\ 1 & -1 & -1 & 0 \end{pmatrix} \quad (7)$$

$$H = \begin{pmatrix} A & B & O \\ C_1 & C_2 & I \end{pmatrix} \quad (8)$$

The systematic codeword vector C consists of the known message binary vector S and the parity binary vector P . Divide $P = (p_1, p_2, \dots, p_{m_b})$ into P_a and P_c as the following:

$$C = (S, P_a, P_c), \quad (9)$$

$$S = (s_1, s_2, \dots, s_{k_b}), \quad (10)$$

$$P_a = (p_{a1}, p_{a2}, p_{a3}, p_{a4}), \quad (11)$$

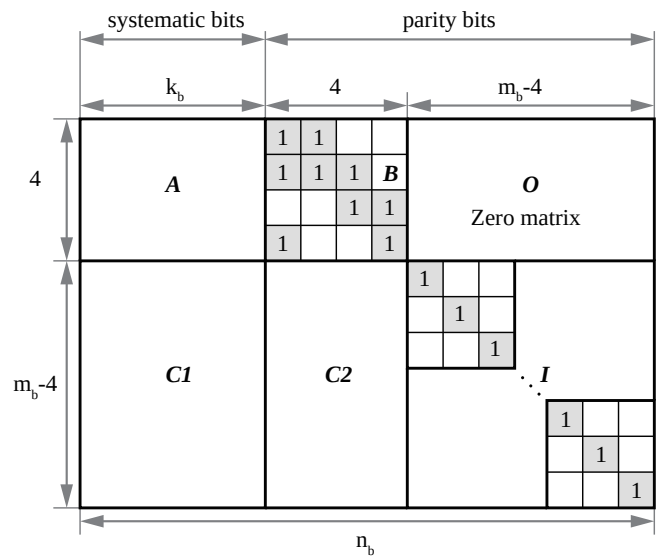


Fig. 3. The structure of an NR QC-LDPC base graph.

$$P_c = (p_{c_1}, p_{c_2}, \dots, p_{c_{m_b-4}}). \quad (12)$$

Each element in S , P_a and P_c denotes a row vector of Z bits. The parity check equation $HC^T = 0^T$ can be written as

$$\begin{pmatrix} A & B & O \\ C_1 & C_2 & I \end{pmatrix} \begin{pmatrix} S \\ P_a \\ P_c \end{pmatrix} = 0^T \quad (13)$$

i.e.

$$AS^T + BP_a^T = 0^T, \quad (14)$$

$$C_1S^T + C_2P_a^T + P_c^T = 0^T. \quad (15)$$

T. T. B. Nguyen et al. propose an efficient encoding algorithm in [9], using two steps to calculate first P_a in (14) and then P_c in (15).

In the first step, write (14) as follows:

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,k_b} \\ a_{2,1} & a_{2,2} & \dots & a_{2,k_b} \\ a_{3,1} & a_{3,2} & \dots & a_{3,k_b} \\ a_{4,1} & a_{4,2} & \dots & a_{4,k_b} \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_{k_b} \end{pmatrix} + \begin{pmatrix} 1 & 0 & -1 & -1 \\ 0 & 0 & 0 & -1 \\ -1 & -1 & 0 & 0 \\ 1 & -1 & -1 & 0 \end{pmatrix} \begin{pmatrix} p_{a_1} \\ p_{a_2} \\ p_{a_3} \\ p_{a_4} \end{pmatrix} = 0. \quad (16)$$

This can be written as the following equations:

$$\sum_{j=1}^{k_b} a_{1,j} s_j + p_{a_1}^{(1)} + p_{a_2} = 0, \quad (17)$$

$$\sum_{j=1}^{k_b} a_{2,j} s_j + p_{a_1} + p_{a_2} + p_{a_3} = 0, \quad (18)$$

$$\sum_{j=1}^{k_b} a_{3,j} s_j + p_{a_3} + p_{a_4} = 0, \quad (19)$$

$$\sum_{j=1}^{k_b} a_{4,j} s_j + p_{a_1}^{(1)} + p_{a_4} = 0. \quad (20)$$

where $p^{(\alpha)} = \alpha p$ denotes the cyclic-shift- α vector of p . By adding up (17) – (20), the following is obtained:

$$p_{a_1} = \sum_{i=1}^4 \sum_{j=1}^{k_b} a_{i,j} s_j. \quad (21)$$

Let λ be the sum of s_j after shift:

$$\lambda_i = \sum_{j=1}^{k_b} a_{i,j} s_j, \text{ for } i = 1, 2, 3, 4, \quad (22)$$

then we get the following from (17)–(20):

$$p_{a_1} = \sum_{i=1}^4 \lambda_i, \quad (23)$$

$$p_{a_2} = \lambda_1 + p_{a_1}^{(1)}, \quad (24)$$

$$p_{a_4} = \lambda_4 + p_{a_1}^{(1)}, \quad (25)$$

$$p_{a_3} = \lambda_3 + p_{a_4} \quad (26)$$

Now we can first calculate λ_i according to (22), then obtain p_{a_i} according to (23)–(26).

In the second step, denote C_1 and C_2 as

$$C_1 = \begin{pmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,k_b} \\ c_{2,1} & c_{2,2} & \dots & c_{2,k_b} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m_b-4,1} & c_{m_b-4,2} & \dots & c_{m_b-4,k_b} \end{pmatrix},$$

$$C_2 = \begin{pmatrix} c_{1,k_b+1} & c_{1,k_b+2} & \dots & c_{1,k_b+4} \\ c_{2,k_b+1} & c_{2,k_b+2} & \dots & c_{2,k_b+4} \\ \vdots & \vdots & \vdots & \vdots \\ c_{m_b-4,k_b+1} & c_{m_b-4,k_b+2} & \dots & c_{m_b-4,k_b+4} \end{pmatrix}. \quad (27)$$

Write (15) as

$$\begin{aligned} p_{c_1} &= \sum_{j=1}^{k_b} c_{1,j} s_j + \sum_{j=1}^g c_{1,k_b+j} p_{a_j}, \\ p_{c_2} &= \sum_{j=1}^{k_b} c_{2,j} s_j + \sum_{j=1}^g c_{2,k_b+j} p_{a_j}, \\ &\vdots \\ p_{c_{m_b-4}} &= \sum_{j=1}^{k_b} c_{m_b-4,j} s_j + \sum_{j=1}^g c_{m_b-4,k_b+j} p_{a_j}. \end{aligned} \quad (28)$$

According to the i -th line of equation, p_{c_i} can be calculated for given C_1, C_2, s_j and p_{a_j} . All the p_{c_i} can be determined in parallel.

V. ENCODING IMPLEMENTATION ON A GPU

In this section, we analyze the parallelism of the encoding algorithm in Section IV and describe the parallel implementation of encoding algorithm above on a CUDA platform.

As mentioned above that only algorithm with high parallelism can be accelerated by GPUs. For a group of computations, as long as any of them does not depend on the result of the others, i.e., there is no data dependency, these calculations can be executed in parallel, such as (2) and (28).

All the binary vectors need to be worked out are λ_i, p_{a_i} ($i = 1, 2, 3, 4$) and p_{c_j} ($j = 1, 2, \dots, m_b - 4$), so the steps that we actually need to executed are (22)–(26) and (28). Obviously there is a sequence between these steps. These steps have to be performed serially. But the Z bits in each of these vectors can be evaluated at the same time. Computing λ and P_c also have parallelism. The procedure is shown as the pseudocode following and illustrated by Fig. 4.

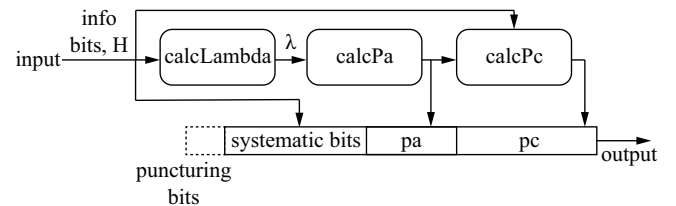


Fig. 4. The kernel functions.

A. Implementation of Encoding one block

After p_{a_i} being obtained, each p_{c_j} in (28) can be calculated simultaneously.

In this procedure, the encoding step is executed on the GPU cores by calling kernel functions. We call three kernel functions, **calcLambda**, **calcPa** and **calcPc**, to calculate parity bits:

- **calcLambda**: Launch $4Z$ threads. In each thread, calculate one bit of λ . In the (i, j) -th thread, loop k_b times. In the k -th loop, obtain the i -th bit of vector $\alpha_{j,k} s_{j,k}$, add it to the j -th bit of vector λ_i .
- **calcPa**: Launch Z threads. In each thread, calculate four bits of $p_{a_1}, p_{a_2}, p_{a_4}$ and p_{a_3} .
- **calcPc**: Launch $(m_b - 4) \times Z$ threads. In the (i, j) -th thread, add up all the $\alpha_{j,k} s_{j,k}$ and $c_{j,k} p_{a_{j,k}}$ bits to obtain $p_{c_{i,j}}$.

We use “ $\langle\langle\langle gridSize, blockSize \rangle\rangle\rangle$ ” to indicate that $gridSize \times blockSize$ threads are launched in the following pseudocode. Inside the kernel function, the index of each thread is $(blockIdx, threadIdx)$. To improve the readability of the pseudocode, we use two-dimensional arrays to locate bits in vectors H, S, λ, Pa and Pc . In the actual implementation, these bits are stored in one-dimensional arrays in order to save the memory space of the GPU. The operator $\oplus$ represents modulo 2 addition.

calcLambda $\langle\langle\langle 4, Z \rangle\rangle\rangle(S, \lambda, H, k_b)$

```

1 temp = 0
2 for i = 1 to  $k_b$ 
3   a =  $H[blockIdx][i]$ 
4   if a != -1
5     temp = temp  $\oplus$   $S[i][(threadIdx + a) \% Z]$ 
6    $\lambda[blockIdx][threadIdx] = temp$ 
```

calcPa $\langle\langle\langle 1, Z \rangle\rangle\rangle(\lambda, P_a)$

```

1 temp =  $\sum_{i=0}^3 \lambda[(threadIdx+1) \% Z][i]$ 
2  $P_a[threadIdx][0] = \sum_{i=0}^3 \lambda[threadIdx][i]$ 
3  $P_a[threadIdx][1] = temp \oplus \lambda[threadIdx][0]$ 
4  $P_a[threadIdx][3] = temp \oplus \lambda[threadIdx][3]$ 
5  $P_a[threadIdx][2] = P_a[threadIdx][3] \oplus \lambda[threadIdx][2]$ 
```

calcPc $\langle\langle\langle m_b - 4, Z \rangle\rangle\rangle(S, P_a, P_c, H, k_b)$

```

1 temp = 0
2 for i = 1 to  $k_b$ 
3   a =  $H[blockIdx][i]$ 
4   if a != -1
5     temp = temp  $\oplus$   $S[i][(threadIdx + a) \% Z]$ 
6 for i = 1 to 4
7   a =  $H[blockIdx+k_b][i]$ 
8   if a != -1
9     temp = temp  $\oplus$   $P_a[i][(threadIdx + a) \% Z]$ 
10  $P_c[blockIdx][threadIdx] = temp$ 
```

As mentioned in Section III that allocating the suitable type of memory for different data can improve efficiency. Our simulation software needs to store information sequences up to Gigabits, which only global memory can accommodate.

Therefore, it is necessary to store the data of information sequences in global memory. The H matrix is an array of constants, so it can be stored in memory that is fast and accessible to all threads. λ is an intermediate variable in the encoding algorithm and can be stored in shared memory. To sum up, we minimize conditional branching and allocate appropriate memory for the data.

B. Complexity

TABLE II
TIME COMPLEXITY OF PROPOSED IMPLEMENTATION

Function	Number of Threads	Operations	Time Complexity
calcLambda	$4Z$	k_b	$O(N)$
calcPa	Z	4	$O(1)$
calcPc	$(m_b - 4)Z$	$k_b + 4$	$O(N)$

Since $k_b \times Z = K = R \times N$, we consider the time complexity of functions **calcLambda** and **calcPc** is $O(N)$. The whole procedure is a linear time implementation $O(N)$.

The space required to encode a (N, K) code block is $K + N - 2Z$ bits in global memory and $4Z$ bits in the shared memory. K bits for input, $N - 2Z$ bits for output and $4Z$ bits for intermediate variables.

VI. RESULTS

We implemented the parallel LDPC encoding algorithm on a Nvidia TITAN RTX GPU with 4608 cores, 1.77GHz core clock rate and 24GB of device memory. Our program is built on top of an Ubuntu 18.04 Linux OS with Nvidia CUDA driver 440.33 and CUDA toolkit release 10.2.

In our experiments, Z is set to 88 and K is set to 1936. We test the bit rate from 1/3 to 8/9. We choose BG1 to construct the parity check matrix H so that $blockSize$ of **calcPc** function from 42 to 0 corresponds to the rate from 1/3 to 8/9. We compare the efficiency when Z is given as an immediate number and when it is transmitted into the kernel function as a parameter.

The throughput of the encoder is measured by dividing the number of the transmitted bits by the duration for the three kernel functions. The information rate is measured by dividing the number of information bits by the duration for the three kernel functions. Fig. 5 shows the Gigabits per second throughput and information rate with $K = 1936$.

It can be seen from Fig. 5 that 38Gbps is the lowest throughput when the bit rate is 1/2 and 62Gbps is the highest when the bit rate is 8/9. 14Gbps is the lowest information rate when the bit rate is 1/3 and 56Gbps is the highest when the bit rate is 8/9. This is expected results because the closer the bit rate is to 1, the fewer bits need to be computed and the more efficient the computation.

Table III compares the performance of our implementation with other implementations. It can be seen from Table III that compared with the implementation on FPGAs [6] [8] and CUDAs [11], the 5G NR LDPC encoder we implemented has a

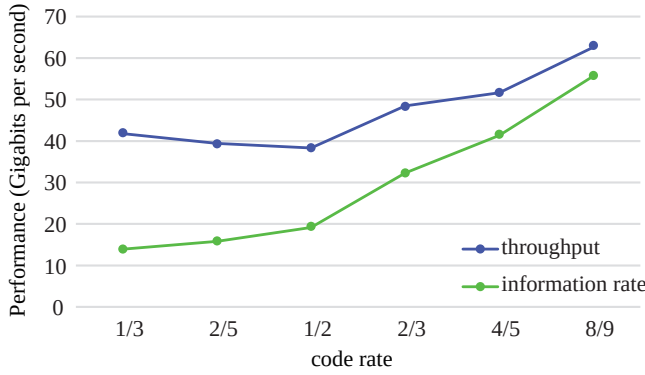


Fig. 5. Throughput and information rate.

higher throughput performance. The throughput implemented in [9] is similar to ours, but the cost in [9] for flexible code rate encoders on the ASIC platform (over a million) is more than hundreds of times that of ours on the CUDA platform (ten thousand). As a result, our implementation combines high performance and flexibility at a reasonable cost.

TABLE III
COMPARISON OF LDPC ENCODERS

Work	Platform	Protocol	Block length	Code rate	Throughput
Proposed	CUDA TITAN RTX	5G NR	5808	1/3	41.8Gbps
			2178	8/9	62.6Gbps
[11]	CUDA TITAN Xp	DVB-S2	16200	1/3	4Gbps
			16200	8/9	20Gbps
[6]	FPGA Xilinx Virtex 4	802.3an	2048	5/6	10Gbps
[8]	FPGA Altera Stratix II	DVB-S2	16200	7/9	19Gbps
[9]	ASIC	5G NR	4224	1/3	43.8Gbps
			2376	8/9	61.4Gbps

VII. CONCLUSION

In this paper, we implement a high throughput LDPC encoding algorithm on CUDA platform. The experimental results show that our LDPC encoding module achieves 38–62Gbps throughput on a single GPU, depends on the bit rate. Compared with the FPGAs and ASICs, our implementation achieve a better trade-off among throughput, flexibility and cost. Therefore, our implementation of LDPC encoder is a good solution satisfying the need of a high performance communication link simulation platform.

ACKNOWLEDGMENT

This project was supported by the National Natural Science Foundation of China (No. 61701484) and Open Research Fund of Key Laboratory of Space Utilization, CAS (No. Y8031931QY).

REFERENCES

- [1] R. G. Gallager, "Low-density parity-check codes," M.I.T. Press, Cambridge, MA, 1963.
- [2] 3rd Generation Partnership Project Technical Specification 38.212 V16.1.0, Mar. 2020.
- [3] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," IEEE Trans. Inf. Theory, vol. 47, no. 2, pp. 638–656, Feb. 2001.
- [4] D. U. Lee, W. Luk, C. Wang, and C. Jones, "A flexible hardware encoder for low-density parity-check codes," in Proc. 12th Annu. IEEE Symp. Field-Programmable Custom Comput. Mach. (FCCM 2004), Apr. 2004, pp. 101–111.
- [5] C. Fewer, M. Flanagan, and A. Fagan, "A versatile variable rate LDPC codec architecture," IEEE Trans. Circuits Syst. I: Reg. Papers, vol. 54, no. 10, pp. 2240–2251, Oct. 2007.
- [6] A. Cohen and K. Parhi, "A low-complexity hybrid LDPC code encoder for IEEE 802.3an (10 GBase-T) Ethernet," IEEE Trans. Signal Process., vol. 57, no. 10, pp. 4085–4094, Oct. 2009.
- [7] A. Mahdi and V. Paliouras, "A low complexity-high throughput QC-LDPC Encoder," IEEE Transactions on Signal Processing, vol. 62, no. 10, May. 2014.
- [8] A. A. A. Hariri, F. Monteiro, L. Sieler and A. Dandache, "A high throughput configurable parallel encoder architecture for quasi-cyclic low-density parity-check codes," On-Line Testing Symposium (IOLTS), 2013 IEEE 19th International, 2013.
- [9] T. T. B. Nguyen, T. N. Tan and H. Lee, "Efficient QC-LDPC encoder for 5G new radio," Electronics, vol. 8, no. 6, pp. 668–, Jun. 2019.
- [10] J. Cheng, M. Grossman and T. McKercher, Professional CUDA C Programming, 1st ed., vol. 1, John Wiley & Sons, Inc., Hoboken, NJ, USA, 2014.
- [11] D. Kun, "High throughput GPU LDPC encoder and decoder for DVB-S2," 2018 IEEE Aerospace Conference, Mar. 2018.



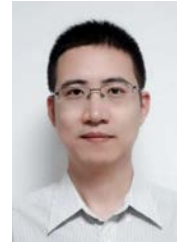
Shixin Liao received her degree of software engineering from Wuhan University, China, in 2019. Now, she is toward her M.Sc degree at University of Chinese Academy of Sciences. Her research activities focus on parallel computing and error control coding.



Yueying Zhan is an associate Research Fellow at Chinese Academy of Science. She received PhD degree at Beijing University of Posts and Telecommunications (BUPT), Beijing, China. Her main research interests include free space optical communication, high speed optical signal processing technologies based on semiconductor optoelectronic devices.



Ziyuan Shi received his degree of communication engineering from Beijing Jiaotong University in 2018. Now, he is toward his Ph.D degree at University of Chinese Academy of Sciences. His research focuses on FSO communication.



Lei Yang received the B.E. and M.Sc. degrees in electronics engineering from Beihang University, Beijing, China, in 2004 and 2007, respectively, and the Ph.D. degree in communications and information system from Beijing Institute of Technology, Beijing, China, in 2015. He received the Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA) in 2019. He is currently a professor at the Technology and Engineering Center for Space Utilization of Chinese Academy of Sciences. His current research interests include free-space optical communications, error control coding, spectrum-efficient coded modulation, and iterative signal processing.