Multimedia Tools and Applications (2020) 79:34287–34300
https://doi.org/10.1007/s11042-020-08738-4

A parallel sliding-window belief propagation algorithm for Q-ary LDPC codes accelerated by GPU

Bowei Shan [1] · Sihua Chen [1] · Yong Fang [1]

Received: 9 August 2019 / Revised: 26 December 2019 / Accepted: 13 February 2020 /
Published online: 4 March 2020

Abstract
In this paper, a parallel Sliding-Window Belief Propagation algorithm to decode Q-ary Low-Density-Parity-Codes is proposed. This algorithm is accelerated by taking advantage of high parallel features of GPU, and applied to video compression under distributed video coding framework. The experiment results show that our parallel algorithm achieves 2.3× to 30.3× speedup ratio under 256 to 2048 codeword length and 69.21× to 78.31× speedup ratio under 16,384 codeword length than sequential algorithm.

Keywords SWBP · LDPC · GPU · DVC

1 Introduction

Distributed Video Coding (DVC) [7] is an advanced asymmetric coding scheme which encodes individual frame independently while decodes them conditionally. This feature makes DVC widely applied in mobile device environment. As a capacity-approaching channel codes, Low-Density-Parity-Codes (LDPC) codes [6, 9] have been used to compress video under DVC framework [15]. LDPC codes were first invented by Gallager [6] in 1962, and rediscovered by MacKay and Neal [9] in 1996. Thereafter, it has become one of the hottest topic for both research and industrial community.

In 1998, MacKay *et al*. [3] generalized the binary LDPC to finite fields GF($Q = 2^q$) and proposed a Q-ary LDPC (QLDPC). QLDPC provides a better choice for practical multimedia problems, *e. g*., video and image compression, as one pixel is normally represented by 8 bits at least. Most decoders of LDPC codes are implemented by Belief Propagation (BP) algorithm [8] (also known as "sum-product" algorithm). The Q-ary BP (QBP) algorithm was used to decode QLDPC. The QBP has a computing complexity $O(NtQ^2)$, where N is the codeword length, and t is the mean column weight of sparse parity check matrix $\mathbf{H}$. To reduce its computing burden,

✉ Yong Fang
fy@chd.edu.cn

[1] School of Information Engineering, Chang'an University, Xi'an 710064, China

Declercq *et al*. [1] proposed a fast Q-ary BP algorithm, whose idea is to replace the convolutional operations by Fast Fourier Transform (FFT) and reduces the complexity to $O(NtQ\log_2(Q))$.

To improve the performance of BP algorithm, Fang [4] presented a Sliding-Window Belief Propagation (SWBP) algorithm, whose idea is to adaptively select optimal local bias probabilities to seed the variable nodes of BP. A lot of experiments [4, 5] showed that SWBP could achieve better performance with less iteration times. In addition, it is very easy to implement and insensitive to the initial settings. Recently, Q-ary SWBP (QSWBP) [14] has been proposed to deal with the QLDPC codes and achieved better performance and robustness, while it still suffers from heavy computing complexity.

Graphics Processing Unit (GPU) [10] invented by NVIDIA, by means of highly parallel structure, has demonstrated powerful ability for high performance computing. Inspired by GPU's amazing ability, we propose a parallel version of QSWBP and accelerate it by GPU. In 2016, the joint-bitplane BP has been accelerated by GPU [2]. In 2019, A parallel binary SWBP algorithm has been accelerated by GPU and obtained remarkable speedup ratio [12]. To our best knowledge, parallel QSWBP algorithm has still not been presented. Instead of C/C++, we will use MATLAB as our programming platform in this paper. As a high-level language for scientific computing and rapid prototyping engineering problems, MATLAB has many advantages, *e. g*.eliminating pointer to avoid memory access error; powerful manipulation of vector and matrix; and concise and efficient vectorization instructions. Since 2010, MATLAB has introduced supports to GPU [11]. We use our parallel QSWBP algorithm to decode a small fractue of video under DVC framework. The numertical experiments are performed to investigate the accelerating effects between parellel and sequence algorithm. A brief version of this paper has been published in IoTaaS 2019 conference [13], and we hereby present the detaild discussions and the application of this algorithm.

2 QSWBP algorithm

2.1 Correlation model

Let $A = [0 : Q)$ denote the alphabet. Let $x, y \in A$ denote the realization of X and Y, which are two random variables. Let X^n be the source to be compressed at the encoder. Let Y^n be the Side Information (SI) that resides only at the decoder. Let $X^n = Y^n + Z^n$. We model the correlation between input X^n and output Y^n as a virtual channel with following properties: Y^n and Z^n are independent with each other; $p_{Z^n}(z^n) = \prod_{i=1}^{n} p_{Z_i}(z_i)$, where $p_X(x)$ denotes the Probability Mass Function (pmf) of discrete random variable X; pmfs of Z_i's may be different, where $i \in [0 : n]$.

We use Truncated Discrete Laplace(TDL) distribution to model Z_i:

$$p_{X_i|Y_i}(x|y) \propto \frac{1}{2b_i}\exp\left(-\frac{|x-y|}{b_i}\right) \tag{1}$$

where b_i is the local scale parameter. Since $\sum_{x=0}^{Q-1} p_{X_i|Y_i}(x|y) = 1$, we can obtain

$$p_{X_i|Y_i}(x|y) = \exp\left(-\frac{|x-y|}{b_i}\right) L_Q(b_i, y) \tag{2}$$

where $L_Q(b,y) = \sum_{x=0}^{Q-1} \exp\left(-|x-y|b_i\right)$. To reduce the computing complexity, we use integration to approximate the summation. When b and Q are reasonably big, this approximation is precise enough by

$$\begin{aligned} L_Q(b,y) &\approx \int_0^{Q-1} \exp\left(-\frac{|x-y|}{b}\right) dx \\ &= 2b\left(1-\frac{1}{2}\exp\left(\frac{y-(Q-1)}{b}\right)-\frac{1}{2}\exp\left(-\frac{y}{b}\right)\right) \end{aligned} \tag{3}$$

2.2 Encoding

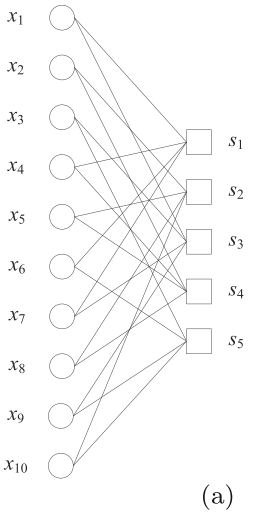
The encoder uses QLDPC codes to compress source $\mathrm{x} \in [0:Q)^n$ to get syndrome $\mathrm{s} \in [0:Q)^n$. This process is performed by matrix-vector multiplication over the finite field GF(Q):

$$\mathrm{s} = \mathbf{H}\mathrm{x} \tag{4}$$

where $\boldsymbol{H} \in [0:Q)^{m \times n}$ is the sparse parity-check matrix. In $\mathbf{H}$, i-th column corresponds to source node x_i, and j-th row corresponds to syndrome node s_i. If the elementary of $\mathbf{H}$ $h_{j,i} \neq 0$, an edge connects s_i and x_i in the bipartite graph of $\mathbf{H}$, as illustrated in Fig. 1. We define the indices of all source nodes that are connected to syndrome node s_j as $N_j =^{\Delta} \leq ft\{i : h_{j,i} \neq 0 \subset [1:n]$, and indices of all syndrome nodes that are connected to source node x_i as $M_i =^{\Delta} \{j : h_{j,i} \neq 0\} \subset [1:m]$.

2.3 Decoding

The decoder seeds source nodes x according to SI y, and runs QBP algorithm to recover x. For the belief propagation between source nodes and syndrome nodes, we give following definitions: $\xi_i(x)$ is intrinsic pmf of source node x_i; $\zeta_i(x)$ is overall pmf of source node x_i; $r_{i,j}(x)$ is the



$$\mathbf{H} = \begin{bmatrix} h_{1,1} & 0 & 0 & h_{1,4} & 0 & h_{1,6} & h_{1,7} & 0 & 0 & 0 \\ 0 & h_{2,2} & 0 & 0 & h_{2,5} & 0 & 0 & h_{2,8} & 0 & h_{2,10} \\ h_{3,1} & 0 & h_{3,3} & 0 & 0 & 0 & h_{3,7} & 0 & h_{3,9} & 0 \\ 0 & h_{4,2} & 0 & h_{4,4} & h_{4,5} & 0 & 0 & h_{4,8} & 0 & 0 \\ 0 & 0 & h_{5,3} & 0 & 0 & h_{5,6} & 0 & 0 & h_{5,9} & h_{5,10} \end{bmatrix}$$

(b)

Fig. 1. (2,4)-regular QLDPC code of length 10. (a) The bipartite graph, where circles represent source nodes and squares represent syndrome nodes; (b) parity check matrix.

pmf passed from source node x_i to syndrome nodes s_j; and $q_{j,\,i}(x)$ is the pmf passed from syndrome nodes s_j to source node x_i, where $j \in M_i$ and $i \in N_j$.

BP includes 5 steps:

1. Initializing BP:

$$\xi_i(x) = \zeta_i(x) = p_{X_i|Y_i}(x|y) \tag{5}$$

$$q_{j,i}(x) = 1/Q \tag{6}$$

where $p_{X_i|Y_i}(x|y)$ is calculated by (3)

2. Source-to-Syndrome BP:

$$r_{i,j}(x) \propto \frac{\zeta_i(x)}{q_{j,i}(x)} \tag{7}$$

3. Syndrome-to-Source BP:

$$q_{j,i}\left(h_{j,i}x\right) = F^{-1}\left\{\frac{\psi_j(w)}{F\left\{r_{i,j}\left(h_{j,i}x\right)\right\}}\right\} \tag{8}$$

where $\psi_j(w) = \prod_{i \in I_j} F\left\{r_{i,j}\left(h_{j,i}x\right)\right\}$ for $j \in [1 : m]$ and $w \in [0 : Q)$, $F\{\cdot\}$ denotes the Fourier Transform, and $F^{-1}\{\cdot\}$ denotes the inverse Fourier Transform.

4. Overall pmf of Source Nodes:

$$\zeta_i(x) = \xi_i(x) \prod_{i \in J_i} q_{j,i}(x) \tag{9}$$

5. Hard Decision and Convergence Test:

$$\widehat{x}_i = \arg\max_{x \in [0:Q)} \zeta_i(x) \tag{10}$$

If $s = \mathbf{H}\hat{x}$, the decoding process finished successfully; otherwise, more iterations need be performed until either $s = \mathbf{H}\hat{x}$ or the iteration times exceed a prespecified threshold.

2.4 SWBP algorithm

In QBP, the source nodes need be seeded with local scale parameter b of virtual correlation channel. In [1, 8], the parameter of virtual correlation channel is estimated by SWBP algorithm. In this paper, we will use expected L_1 distance between each source symbol and its corresponding SI symbol defined as

$$\mu_i \triangleq \sum_{x=1}^{Q} (\zeta_i(x) \cdot |x - y_i|) \tag{11}$$

Then, the estimated local scale parameter $\widehat{b}$ is calculated by averaging the expected L_1 distances of its neighbors in a window with size-$(2\eta + 1)$

$$\widehat{b}_i(\eta) = \frac{t_i(\eta) - \mu_i}{min\,(i+\eta, n) - max\,(1, i-\eta)} \tag{12}$$

where

$$t_i(\eta) \triangleq \sum_{i'=\max\,(1,i-\eta)}^{\min\,(i+\eta,n)} \mu_{i'} \tag{13}$$

To calculate (13), we first calculate $t_1(\eta) = \sum_{i'=1}^{1+\eta} u_{i'}$. Then for $i \in [2 : n]$,

$$t_i(\eta) = \begin{cases} t_{i-1}(\eta) + \mu_{i+\eta}, & i \in [2 : (\eta+1)] \\ t_{i-1}(\eta) + \mu_{i+\eta} - \mu_{i-1-\eta}, & i \in [(\eta+2) : (n-\eta)] \\ t_{i-1}(\eta) + \mu_{i-1-\eta}, & i \in [(n-\eta+1) : n] \end{cases} \tag{14}$$

Same as [1, 8], the main purpose of QSWBP is to find a best half window size $\widehat{\eta}$. We define an expected rate:

$$\begin{aligned} \gamma(\eta) &\triangleq -\sum_{i=1}^{n} \sum_{x=0}^{Q-1} \zeta_i(x) \cdot \ln \frac{exp\left(-|x - y_i|\widehat{b}_i(\eta)\right)}{L_Q\left(\widehat{b}_i(\eta), y_i\right)} \\ &= \sum_{i=1}^{n} \left(\ln L_Q\left(\widehat{b}_i(\eta), y_i\right) + \frac{\mu_i}{\widehat{b}_i(\eta)} \right) \end{aligned} \tag{15}$$

where $L_Q\left(\widehat{b}_i(\eta), y_i\right)$ is defined by (3). The best half window size $\widehat{\eta}$ is chosen by

$$\widehat{\eta} = \arg\min_{\eta} \gamma(\eta), \tag{16}$$

It is a natural idea that best half window size should minimize the expected rate. The flowchart of QSWBP algorithm is illustrated in Fig. 2.

3 Parallel QSWBP algorithm

3.1 Complexity analysis

In Fig. 2, the decoding process includes 4 parts.

- The Source-to-Syndrome BP step needs n times iterations to calculate $r_{i,j}(x)$ Since there are n source nodes.
- The Syndrome-to-Source BP step needs m times iterations to calculate $q_{j,i}(x)$ since there are m syndrome nodes. Let rw_j denotes the j-th row weight, in each iteration, the Fourier Transform needs be calculated for rw_j times and inverse Fourier Transform also needs be calculated for rw_j times. We use Fast Fourier Transform(FFT) to implement Fourier Transform and inverse Fourier Transform. If the codeword length is n, each FFT needs $n\log_2(n)$ real multiplies and real adds.
- The Computing Overall pmf step needs n times iterations to calculate $\zeta_i(x)$.
- In SWBP step, to find the best half window size, a search strategy [5] was proposed that only searches half window size from $\eta \in \left\{1^2, \ldots, \left\lfloor \sqrt{\frac{n-1}{2}} \right\rfloor^2\right\}$. Although this strategy could remarkably reduce the searching iterations, we found that the best half window size might be omitted according to our experiment results. Therefore, we will evaluate all expected rates from $\eta \in \left\{1, 2, \ldots, \left\lfloor \frac{n-1}{2} \right\rfloor\right\}$ which needs $\left\lfloor \frac{n-1}{2} \right\rfloor$ iterations. In practice, since $\gamma(1)$ is

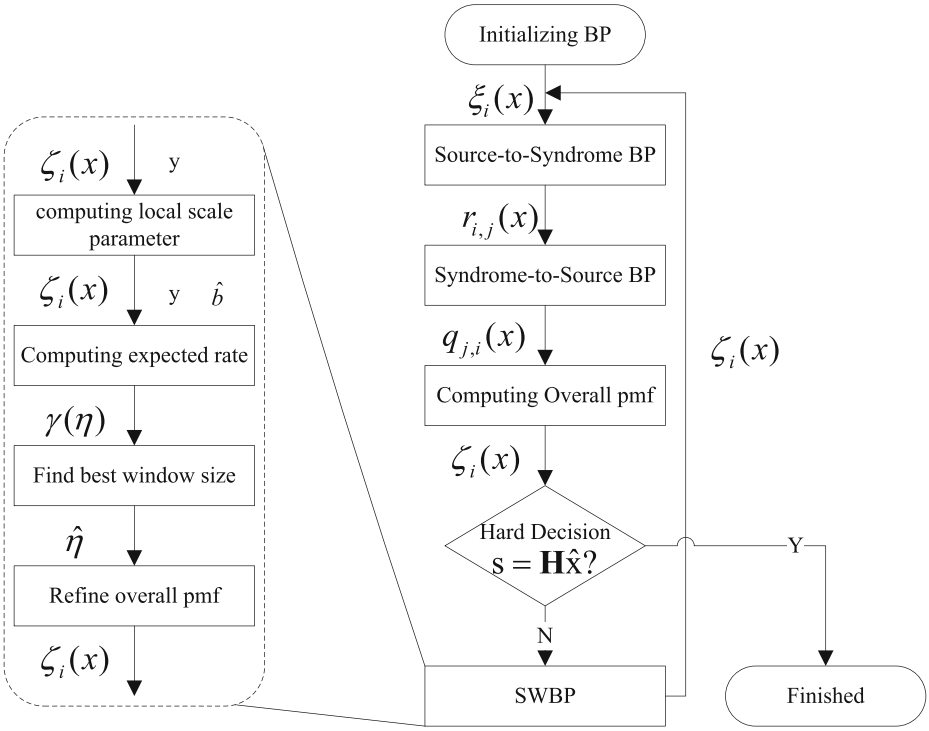


Fig. 2 Flowchart of QSWBP

always big enough, it is obviously unnecessary to calculate it while $\eta = 1$. In each SWBP iteration, getting $t_i(\eta)$ needs n iterations in (14) and getting $\gamma(n)$ also needs n iterations in (15) in which the ln functions is very time consuming.

Based on above analysis, we conclude that syndrome-to-source BP step and SWBP step are two major time consuming parts in the entire algorithm. To verify our conclusion, we use the profile function in MATLAB to investigate the details of time consuming. Profile function [10] could record the executive time of each function in MATLAB code. The time consuming result is shown in Fig. 3, where *qldpc _ test* is the main function which costs 5.548 s, *ntt* is the FFT function which is called for 8192 times and totally cost 3.912 s, and *swbp _ lap* is the SWBP function which is called for 7 times and totally cost 1.607 s. Profile analysis means FFT and SWBP are two major bottlenecks which agrees well with ours analysis. Since one GPU has large amount of cores, which can run many threads simultaneously, we take advangtage of this feature of GPU to accelerate above bottlenecks. In the our parallel algorithm, the bottleneck is divided into many pieces, which has no corelation with each others, and each piece can run on one core of GPU. The details of parallel algorithm are introduced in next two subsections.

3.2 Parallel syndrome-to-source BP algorithm

The sequential Syndrome-to-Source BP algorithm needs m iterations to calculate $q_{j,\ i}(x)$ as there are m syndrome nodes. Since these m iterations are independent with each other, they can be calculated in parallel. We take Fig. 1(b) as an example. Figure 4 depicts our parallel algorithm, where the number in square means the row position of non-zero elements in parity-

Profile Summary

Generated 12-Jul-2018 13:28:53 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
qldpc_test	1	5.548 s	0.007 s	
dec	1	5.450 s	0.194 s	
ntt	8192	3.478 s	3.478 s	
swbp_lap	7	1.607 s	0.011 s	
swbp_lap>getb	868	1.596 s	1.596 s	
enc	9	0.209 s	0.023 s	
gf.mtimes	1153	0.098 s	0.060 s	
get_pmf_lap	8	0.026 s	0.026 s	
gfadd	1	0.010 s	0.001 s	
loadTG	1	0.007 s	0.007 s	
num2str	3	0.003 s	0.002 s	
nttsft	1	0.002 s	0.002 s	
randl	1	0.002 s	0.002 s	
int2str	3	0.001 s	0.001 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

Fig. 3 Time consuming details of sequantial Q-ary SWBP

check-matrix **H** of Fig. 1(b). 5 threads run simultaneously on GPU to calculate each $q_{j,i}(x)$ and each thread calculates FFT in sequence.

3.3 Parallel SWBP algorithm

In sequential SWBP algorithm, each window size setup iteration could generate an expected rate $\gamma(\eta)$, which is calculated by (15). Any two expected rate $\gamma(\eta_1)$ and $\gamma(\eta_2)$ $(\eta_1 \neq \eta_2)$ are uncorrelated, and can be computed in parallel. In our parallel algorithm, all $\gamma(\eta)$, $\eta \in \left\{1, 2, \ldots, \left\lfloor \frac{n-1}{2} \right\rfloor \right\}$ would be calculated simultaneously by thousands of threads on GPU. Once $\gamma(\eta)$, $\eta \in \left\{1, 2, \ldots, \left\lfloor \frac{n-1}{2} \right\rfloor \right\}$ was obtained, we could use *min()* function in MATLAB to get the smallest γ and corresponding best η from array $\gamma(\eta)$. The sequential and parallel algorithm are illustrated in Fig. 5.

3.4 Vectorization

Thanks to the features of MATLAB language, we could manipulate matrix and vector by vectorizing code instead of loop-based code. We take calcualting (11) for example. Listing 1 is our MATLAB implementation of (11) by loop-based code and vectorizing code, repectively. From these codes, we could find vectorization has three advantages: less code means less error; it's easier to understand since vectorizing code appears like the mathematical expressions in equations; and it runs much faster than loop-based code since MATLAB is optimized for vectorization. By using vectorization, the code appears more concise and elegents. Our vectorized source code has reduced 10% length than loop-based code.

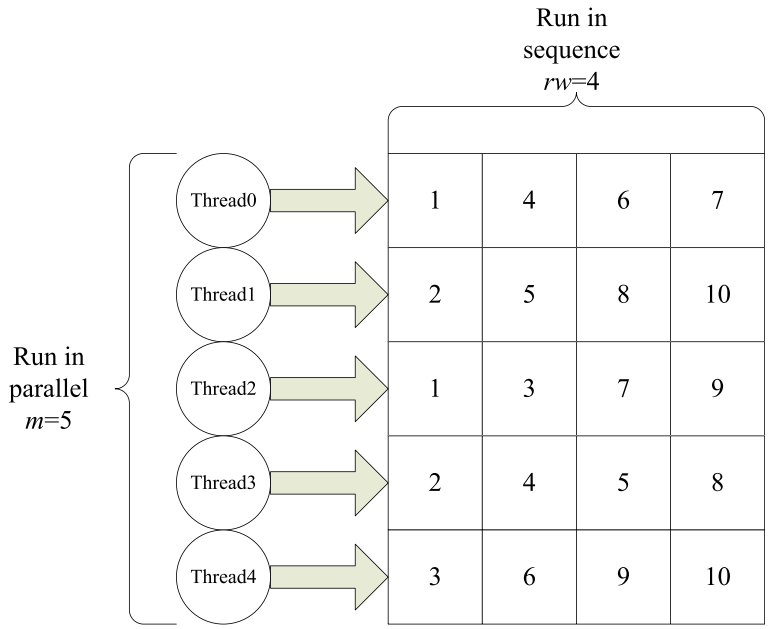


Fig. 4 Parallel Syndrome-to-Source BP algorithm

```
% this is loop-based code
mu=0;
for x=1:Q
    mu=mu+zeta(x)*(abs(x-y));
end

% this is corresponding vectorizing code
x=1:Q;
mu=sum(zeta(x).*(abs(x-y)))
```

Listing 1 MATLAB code of (11)

4 Video compression by QLDPC

Under DVC framework, the encoder is implemented by Pixel-Domain and Transform-Domain (PDTD) scheme. PDTD divides video frames into Key Frame, which is encoded and decoded using a conventional intraframe codec, and Wyner-Ziv Frames. A block-wise DCT is performed for Wyner-Ziv frames to obtain the transform coefficients X^{DCT}, which are independently quantized and grouped into coefficient bands. These bands are compressed by LDPC encoder. At the decoder, the correlation between X^{DCT} and side information S^{DCT} is modeled as a Laplacian distribution. LDPC decoder reconstructs the coefficient bands with the corrsponding side information and performs a inverse DCT to generate the reconstructed Wyner-Ziv frames.

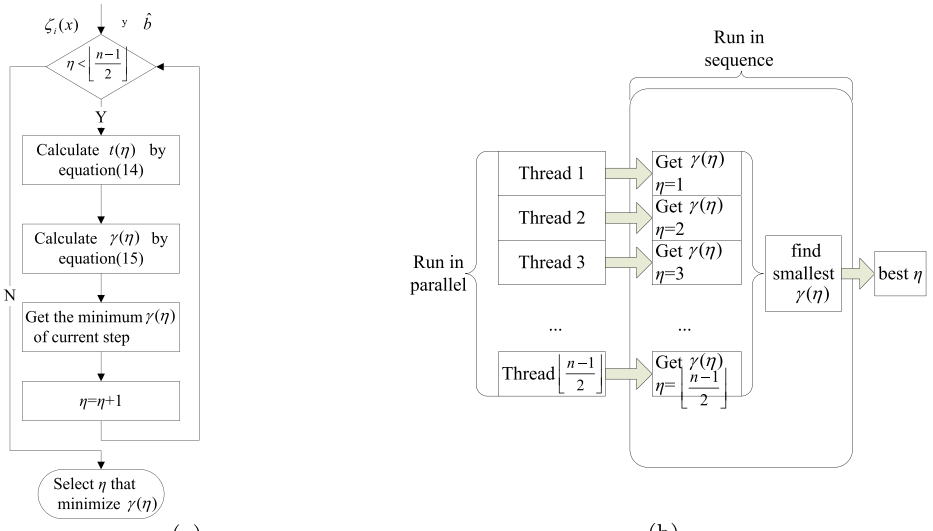


Fig. 5. (a) sequential SWBP algorithm, (b) parallel SWBP algorithm.

One frame in the video is normally represented by an $n \times n$ 2D source. Let $x^{n,n}$ and $y^{n,n}$ be two $n \times n$ 2D sources, where

$$x^{n,n} \triangleq \begin{pmatrix} x_{0,0} & \cdots & x_{0,n-1} \\ \vdots & \ddots & \vdots \\ x_{n-1,0} & \cdots & x_{n-1,n-1} \end{pmatrix} \tag{17}$$

where $x_{i,j} \in [0 : Q)$. The correlation between them follows the setup in Section 2.1.

At the encoder, $x^{n,n}$ is first vectorized into an Q-ary temporary vector $v^{n,n}$. Then $v^{n,n}$ is performed a matrix-vector multiplication over the finite field GF(Q) to compress source to syndrome s^m:

$$s^m = \boldsymbol{H} v^{n,n}. \tag{18}$$

where $\mathbf{H}$ is an $m \times (n \times n)$ sparse parity-check matrix.

At the decoder, the QSWBP is performed with the help of syndrome and side information to recover the source.

5 Experiment results

In our experiments, we use Intel Core i7 with 3.60Ghz as our CPU and NVIDIA GTX 1080Ti as our GPU. The detailed parameters of this GPU are listed in Table 1. We used MATLAB 2014b as our development platform, since this version provides fully support to GPU acceleration.

5.1 Performance of parallel QSWBP

To evaluate the performance of our parallel algorithm, we perform two experiments with different Q.

In first experiment, we set $Q = 256$, and use 4 different regular LDPC codes as our input. The parameters of these LPDC codes are listed in Table 2. To eliminate the random errors, we perform 100 tests and average these outputs as our final results. The experiment result is illustrated in Fig. 6, which shows that parallel QSWBP algorithm achieves 2.9× to 30.3×

Table 1 Parameters of GPU platform

Item	Property
Name	NVIDIA GTX 1080Ti
Core Name	Pascal
Frequency	1.58 GHz
Technology	16 nm
CUDA capability version:	6.1
CUDA cores:	3584
SMs:	28
Threads per block:	1024
Each dimension of a block:	1024×1024×64
Warp size:	32
Constant memory:	64 KB
Shared memory per block:	48 KB
Memory:	11GB GDDR5

Table 2 Different LDPC code parameters(N is codeword length, K is information bit number)

Test	1	2	3	4
N	256	512	1024	2048
K	128	256	512	1024
Maximum degree	4	4	4	4
Code Type	regular	regular	regular	regular

accelerating ratio than sequential QSWBP algorithm. The longer the codeword length, the higher the accelerating ratio.

The second experiment uses first experiment's LDPC codes as input, while we set Q = 2048. The experiment result is illustrated in Fig. 7, which shows that parallel QSWBP algorithm achieves 2.3× to 11.4× accelerating ratio than sequential QSWBP algorithm. The trend of accelerating ratio is same with the first one.

Normally, successful decoding needs 11 rounds BP iterations under Q = 256, but only 3 rounds BP iterations under Q = 2048. As a result, the totally running time of second experiment is shorter than that of first experiment. Because of the accumulating effects, first experiment achevieves higher accelerating performance than second experiment does.

5.2 Performance of video decoding

We borrow a YUV video sequence named *Foreman* from [16], and choose first 4 frames from *Foreman* as the source. Each frame is cropped to the size of 128×128 pixels. We construct a regular LDPC code with condeword length 16,384, and information bit number 8192. Then the rate is 1/2. The alphabet cardinality Q is fixed to $2^8 = 256$.

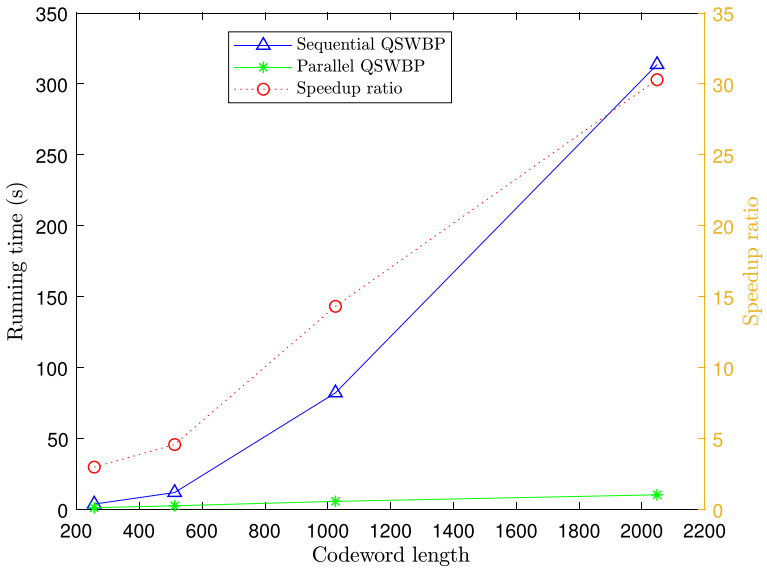


Fig. 6 Running time and speedup ratio under Q = 256

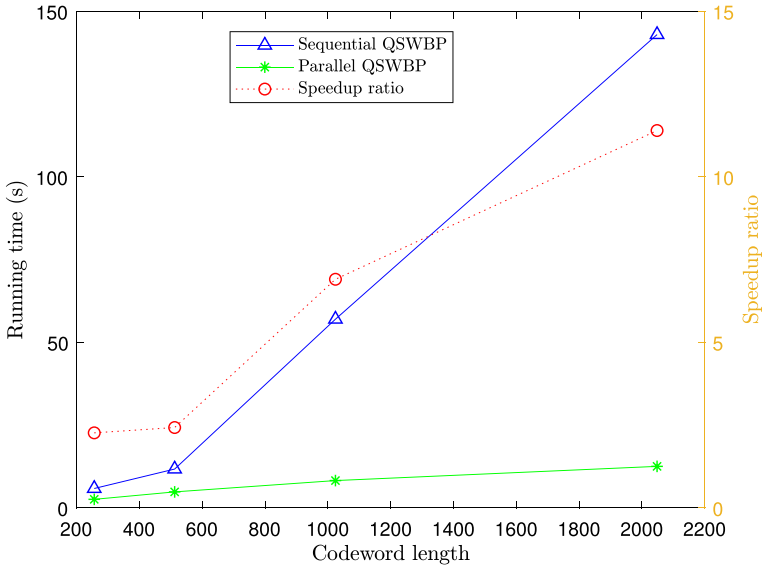


Fig. 7 Running time and speedup ratio under $Q = 2048$

Our early experiments have demonstrated that QSWBP outperforms QBP in video decoding [14]. In this paper, only the parallel and sequential QSWBP are evaluated by decoding the *Foreman* video. The running time is used as the metric to evaluate the performance of two algorithms. To eliminate the random errors, we perform 100 decoding processes and average these running times as our final results. The experimental results are listed in Table 3. Both parallel and sequental QSWBP obtain the same recovered frames which are displayed in Fig. 8.

From Table 3, we can find that, under different frames, parallel QSWBP achieved 69.21× to 78.31× speedup ratio. It is because that a very long LDPC code length (16384) need more iterations to search a best window size in the sequential QSWBP.

6 Conclusion

A parallel Q-ary SWBP algorithm to decode regular LDPC codes with different codelength and Q value has been proposed. We accelerate this algorithm with GPU and MATLAB

Table 3 Performance Comparision Between Parallel and Sequential QSWBP Algorithms

#Frame	Running Time (s)		Speedup Ration
	Sequential QSWBP	Parallel QSWBP	
1	156,852.39	78.31	2002.76
2	149,725.45	69.21	2163.19
3	150,328.21	70.12	2143.74
4	151,236.12	71.25	2122.37

Fig. 8 Recovered 4 Frames of *Foreman*

vectorization technique. Experiment results show that parallel algorithm achieves 2.9× to 30.3× speedup ratio under $Q = 256$ and 2.3× to 11.4× speedup ratio under $Q = 2048$. The video decoding experiment shows that parallel algorithm achieves 69.21× to 78.31× speedup ratio under $Q = 256$. In our future work, we will implement the parallel algorithm on the FPGA platform to extend its applications.

Acknowledgements We would like to thank colleagues in Chang'an University for their helpful discussions.

References

1. Barnault L, Declercq D (2003) Fast decoding algorithm for LDPC over GF(2^q). In: Information Theory Workshop, 2003. Proceedings. 2003 IEEE, pp. 70–73. IEEE
2. Dai Y, Fang Y, Yang L, Jeon G (2016) Graphics processing unit-accelerated joint-bitplane belief propagation algorithm in DSC. J Supercomput 72(6):2351–2375
3. Davey MC, MacKay D (1998) Low-density parity check codes over GF(q). IEEE Commun Lett 2(6):165–167
4. Fang Y (2012) LDPC-based lossless compression of nonstationary binary sources using sliding-window belief propagation. IEEE Trans Commun 60(11):3161–3166
5. Fang Y (2013) Asymmetric slepian-wolf coding of nonstationarily-correlated m-ary sources with sliding-window belief propagation. IEEE Trans Commun 61(12):5114–5124
6. Gallager R (1962) Low-density parity-check codes. IRE Transactions on information theory 8(1):21–28
7. Girod B, Aaron AM, Rane S and Rebollo-Monedero D (2005) Distributed Video Coding, in Proceedings of the IEEE, vol. 93(1), pp. 71–83

8. MacKay DJC (1999) Good error-correcting codes based on very sparse matrices. IEEE Trans Inf Theory 45(2):399–431
9. Mackay DJC, Neal RM (1997) Near shannon limit performance of low density parity check codes. Electron Lett 32(6):457–458
10. NVIDIA: http://www.nvidia.com/object/what-is-gpu-computing.html
11. Ploskas N, Samaras N (2016) GPU programming in MATLAB. Morgan Kaufmann
12. Shan B, Fang Y. A GPU accelerated sliding-window belief propagation parallel algorithm for LDPC code. International Journal of Parallel Programming (in press)
13. Shan B et al. (2019) Accelerating Q-ary Sliding-Window Belief Propagation algorithm with GPU. 5th EAI International Conference on IoT as a Service (IoTaas 2019), Xi'an
14. Shan, B. et al. Joint source-channel estimation via sliding-window belief propagation. IEEE Transactions on Wireless Communications (in preparation)
15. Xu Q, Xiong Z (2006) Layered Wyner-Ziv video coding. IEEE Trans Image Process 15(12):3791–3803
16. YUV Video Sequences: http://trace.eas.asu.edu/yuv/index.html