Accelerating Q-ary Sliding-Window Belief Propagation Algorithm with GPU

Bowei Shan(✉), Sihua Chen, and Yong Fang

School of Information Engineering, Chang'an University, Xi'an, China
{bwshan,fy}@chd.edu.cn,1543275321@qq.com

Abstract. In this paper, we present a parallel Sliding-Window Belief Propagation algorithm to decode Q-ary Low-Density-Parity-Codes. The bottlenecks of sequential algorithm are carefully investigated. We use MATLAB platform to develop the parallel algorithm and run these bottlenecks simultaneously on thousands of threads of GPU. The experiment results show that our parallel algorithm achieves 2.3× to 30.3× speedup ratio than sequential algorithm.

Keywords: SWBP · LDPC · GPU · MATLAB

1 Introduction

As a very good error-correcting code [1], Low-Density-Parity-Codes (LDPC) codes are wildly used in Fifth Generation (5G) telecommunication and Internet of Things as s Service (IoTaaS). It was first invented by Gallager [2] in 1962, and unfortunately ignored by information society for more than 30 years. The renaissance of LDPC was triggered by MacKay and Neal in 1996.

Originally, binary LDPC has been decoded by belief propagation (BP) algorithm (also known as "sum-product" algorithm) [1]. In 1998, MacKay *et al.* [3] generalized the binary LDPC to finite fields GF ($Q = 2^q$) and proposed a Q-ary LDPC. The Q-ary BP algorithm is used to decode Q-ary LDCP. To improve the performance of BP algorithm, Fang presented a Sliding-Window Belief Propagation (SWBP) algorithm. A lot of experiments [6] show that SWBP achieves better performance with less iteration times. In addition, it is very easy to implement and insensitive to the initial settings. Incorporating fast Q-ary BP with SWBP is a nature way to attain better performance and robustness, while it still suffers from heavy computing complexity.

Invented by NVIDIA, Graphics Processing Unit (GPU) [7] has demonstrated powerful ability for general-purpose computing. NVIDIA also presented a C-like languages interface named Compute Unified Device Architecture (CUDA) which is a useful tool for researchers to develop the parallel algorithms. Inspired by GPU's amazing ability, we propose a parallel algorithm of Q-ary SWBP and accelerate it by GTX 1080Ti. Although we has used GPU to accelerate parallel

B. Li et al. (Eds.): IoTaaS 2019, LNICST 316, pp. 3–8, 2020.
https://doi.org/10.1007/978-3-030-44751-9_1

binary SWBP algorithm developed by CUDA C++ [8], to our best knowledge, parallel Q-ary SWBP algorithm has still not been presented.

The rest of this paper is organized as: Sect. 2 describes the sequential Q-ary SWBP algorithm. Section 3 analyses the bottlenecks of sequential algorithm and presents the parallel algorithm. Section 4 uses GPU to accelerate the parallel algorithm and gives the experiment results. We concludes this paper in Sect. 5.

2 Q-ary SWBP Algorithm

2.1 Correlation Model

Let $\mathcal{A} = [0 : Q)$ denote the alphabet. Let $x, y \in \mathcal{A}$ 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 three properties:

(1) Additive: Y^n and X^n are independent with each other;
(2) Memoryless: $p_{Z^n}(z^n) = \prod_{i=1}^{n} p_{Z_i}(z_i)$, where $p_X(x)$ denotes the Probability Mass Function (pmf) of discrete random variable X;
(3) Nonstationary: 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

$$p_{X_i|Y_i}(x|y) = \exp\left(-\frac{|x-y|}{b_i}\right) \Big/ \mathcal{L}_Q(b_i, y) \tag{2}$$

where $\mathcal{L}_Q(b, y) = \sum_{x=0}^{Q-1} \exp(-|x-y|/b_i)$. 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

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

The encoder uses Q-ary LDPC codes to compress source x $\in [0 : Q)^n$ to get syndrome s $\in [0 : Q)^n$. The decoder seeds source nodes x according to SI y, and runs Q-ary BP 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 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 $j \in \mathcal{M}_i$ and $i \in \mathcal{N}_j$. The encoding and decoding process of $Q - ary$ BP has been stated in [6], and we omit it in this paper.

2.2 SWBP Algorithm

In Q-ary BP, the source nodes need be seeded with local scale parameter b of virtual correlation channel. In [4] and [5], 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

$$\mu_i \triangleq \sum_{x=1}^{Q} (\zeta_i(x) \cdot |x - y_i|) \tag{4}$$

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

$$\hat{b}_i(\eta) = \frac{t_i(\eta) - \mu_i}{\min(i + \eta, n) - \max(1, i - \eta)} \tag{5}$$

where

$$t_i(\eta) \triangleq \sum_{i'=\max(1,i-\eta)}^{\min(i+\eta,n)} \mu_{i'} \tag{6}$$

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

$$t_i(\eta)= \begin{cases} t_{i-1}(\eta) + \mu_{i+\eta}, & i \in [2 : (\eta + 1)] \\ t_{i-1}(\eta) + \mu_{i+\eta} - \mu_{i-1-\eta}, & i \in [(\eta + 2) : (n - \eta)] \\ t_{i-1}(\eta) + \mu_{i-1-\eta}, & i \in [(n - \eta + 1) : n] \end{cases} \tag{7}$$

Same as [4] and [5], the main purpose of SWBP is to find a best half window size $\hat{\eta}$. We define an expected rate:

$$\begin{aligned} \gamma(\eta) &\triangleq - \sum_{i=1}^{n} \sum_{x=0}^{Q-1} \zeta_i(x) \cdot \ln \frac{\exp(-|x - y_i| \big/ \hat{b}_i(\eta))}{\mathcal{L}_Q(\hat{b}_i(\eta), y_i)} \\ &= \sum_{i=1}^{n} \left(\ln \mathcal{L}_Q(\hat{b}_i(\eta), y_i) + \frac{\mu_i}{\hat{b}_i(\eta)} \right) \end{aligned} \tag{8}$$

where $\mathcal{L}_Q(\hat{b}_i(\eta), y_i)$ is defined by (3). The best half window size $\hat{\eta}$ is chosen by

$$\hat{\eta}= \arg\min_{\eta} \gamma(\eta), \tag{9}$$

It is a natural idea that best half window size should minimize the expected rate. The flowchart of Q-ary SWBP algorithm was illustrated in Fig. 1.

3 Parallel SWBP Algorithm

In sequential SWBP algorithm, each window size setup iteration generates 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

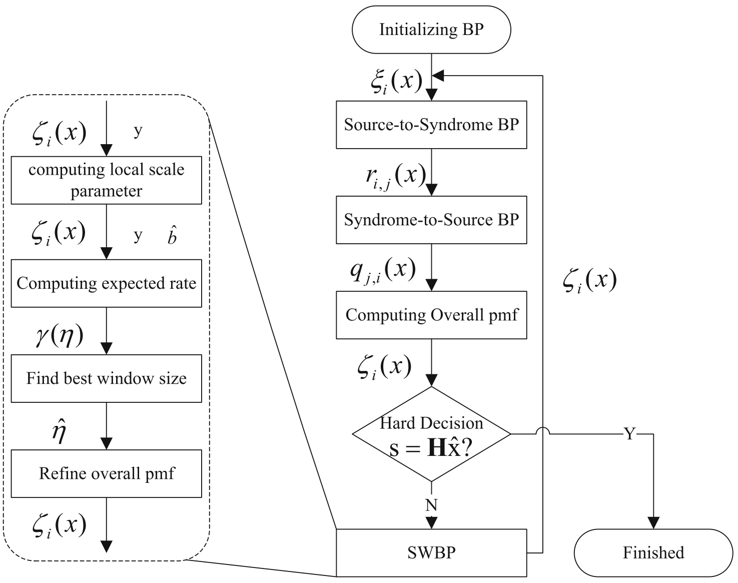

Fig. 1. Flowchart of Q-ary SWBP

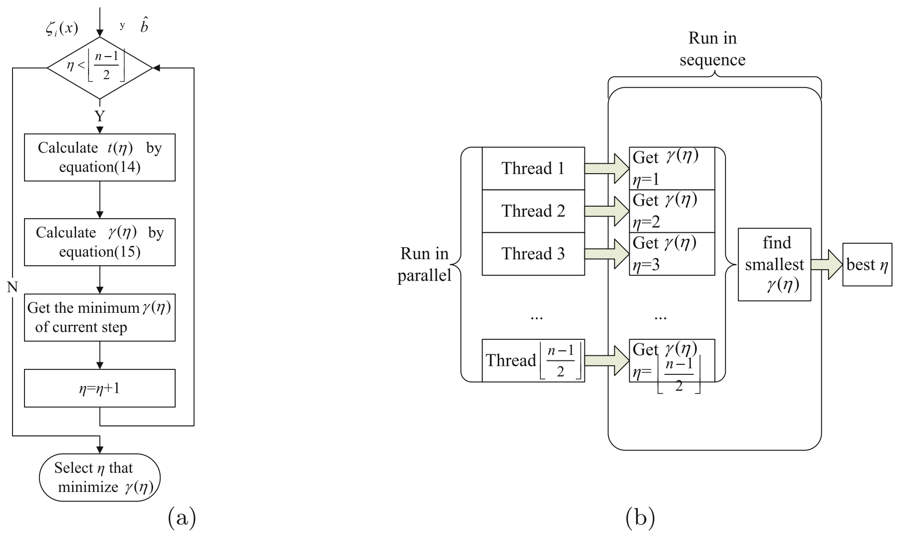
$\zeta_i(x)$ y $\hat{b}$
$\eta < \left\lfloor \frac{n-1}{2} \right\rfloor$
Y
N
Calculate $t(\eta)$ by equation(14)
Calculate $\gamma(\eta)$ by equation(15)
Get the minimum $\gamma(\eta)$ of current step
η=η+1
Select η that minimize $\gamma(\eta)$
(a)
Run in sequence
Run in parallel
Thread 1
Thread 2
Thread 3
...
Thread $\left\lfloor \frac{n-1}{2} \right\rfloor$
Get $\gamma(\eta)$ η=1
Get $\gamma(\eta)$ η=2
Get $\gamma(\eta)$ η=3
...
Get $\gamma(\eta)$ η= $\left\lfloor \frac{n-1}{2} \right\rfloor$
find smallest $\gamma(\eta)$
best η
(b)

Fig. 2. (a) Sequential SWBP algorithm, (b) parallel SWBP algorithm parallel algorithm, all $\gamma(\eta)$, $\eta \in \left\{1, 2, \ldots, \left\lfloor \frac{n-1}{2} \right\rfloor\right\}$ are calculated simultaneously by thousands of threads on GPU. Once $\gamma(\eta)$, $\eta \in \left\{1, 2, \ldots, \left\lfloor \frac{n-1}{2} \right\rfloor\right\}$ are obtained, we 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. 2.

4 Experiment Results

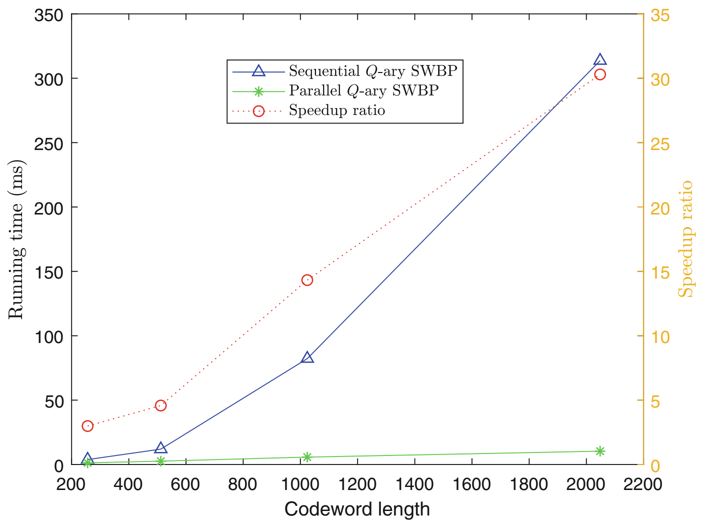

Fig. 3. Running time and speedup ratio under Q=256

In our experiments, we use Intel Core i7 with 3.60Ghz as our CPU and NVIDIA GTX 1080Ti as our GPU. We use MATLAB 2014b as our development platform. We perform numerical experiments to evaluate the performance of our parallel algorithm.

We set Q=256, and use 4 different regular LDPC codes as our input. The parameters of these LPDC codes are listed in Table 1. To eliminate the random errors, we perform 100 tests and average these outputs as our final results. The experiment result is illustrated in Fig. 3, which shows that parallel Q-ary SWBP algorithm achieves 2.9× to 30.3× accelerating ratio than sequential Q-ary SWBP algorithm. The longer the codeword length, the higher the accelerating ratio.

Table 1. 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

5 Conclusion

We propose a parallel Q-ary SWBP algorithm to decode regular LDPC codes with different codelength and Q value. This algorithm can address the bottlnecks of sequential Q-ary SWBP and be accelerated by GPU. Experiment results show that parallel algorithm achieves 2.9× to 30.3× speedup ratio under Q=256. The longer codeword length leads to higher speedup ratio.

References

1. MacKay, D.J.C.: Good error-correcting codes based on very sparse matrices. IEEE Trans. Inf. Theory **45**(2), 399–431 (1999)
2. Gallager, R.: Low-density parity-check codes. IRE Trans. Inf. Theory **8**(1), 21–28 (1962)
3. Davey, M.C., MacKay, D.: Low-density parity check codes over GF(q). IEEE Commun. Lett. **2**(6), 165–167 (1998)
4. Fang, Y.: LDPC-based lossless compression of nonstationary binary sources using sliding-window belief propagation. IEEE Trans. Commun. **60**(11), 3161–3166 (2012)
5. Fang, Y.: Asymmetric Slepian-Wolf coding of nonstationarily-correlated M-ary sources with sliding-window belief propagation. IEEE Trans. Commun. **61**(12), 5114–5124 (2013). https://doi.org/10.1109/TCOMM.2013.111313.130230
6. Fang, Y., Yang, Y., Shan, B., Stankovic, V.: Joint source-channel estimation via sliding-window belief propagation. IEEE Trans. Commun. (2019, submited)
7. NVIDIA. http://www.nvidia.com/object/what-is-gpu-computing.html
8. Shan, B., Fang, Y.: GPU accelerated parallel algorithm of sliding-window belief propagation for LDPC codes. Int. J. Parallel Program. (2019). https://doi.org/10.1007/s10766-019-00632-3