GPU-BASED IMPLEMENTATION OF BELIEF PROPAGATION DECODING FOR POLAR CODES

Zhanxian Liu[1]*, Rongke Liu*[1]*, Zhiyuan Yan*[2]*, Ling Zhao*[1]

[1]School of Electrical and Information Engineering, Beihang University, Beijing, China
[2]Department of Electrical and Computer Engineering, Lehigh University, Pennsylvania, USA

ABSTRACT

Belief Propagation (BP) decoding provides soft outputs and features high-level parallelism. In this paper, we propose an optimized software BP decoder for polar codes on graphics processing units (GPUs). A full-parallel decoding architecture for codes with length $n \leq 2048$ is presented to simultaneously update $n/2$ processing elements (PEs) within each stage and achieve high on-chip memory utilization by using 8-bit quantization. And, for codes with length $n > 2048$, a partial-parallel decoding architecture is proposed to partly update PEs of each stage in parallel and coalesced global memory accesses are performed. Experimental results show that, with incorporation of the G-matrix based early termination criterion, more than 1 Gbps throughput for codes $n \leq 1024$ can be achieved on NVIDIA TITAN Xp at 5 dB while the decoding latency is less than 1 *ms*. Compared with the state-of-the-art works, the proposed decoder achieves throughput speedups from 2.59x to 131x and provides good tradeoff between error performance and throughput.

Index Terms— Polar code, CUDA, SIMT, parallel decoding, GPU

1. INTRODUCTION

Polar codes, originally introduced by Arikan [1], are the first family of theoretically capacity-achieving codes and are regarded as a significant breakthrough in coding theory. Successive-Cancellation (SC) decoding and Belief-Propagation (BP) decoding are two popular decoding algorithms for polar codes. SC decoding features low computational complexity and memory consumption, but performs worse in terms of error-correction capability at short to moderate code lengths. Based on SC decoding, SC list (SCL) decoding aided by Cyclic Redundancy Check (CRC) [2] is proposed to enhance error-correction performance at the expense of increased complexity. However, the type of SC-based decoding approaches characterizes serial message updating and thus suffers long decoding latency and low decoding throughput.

The work was supported by the National Natural Science Foundation of China (61871009).

Alternatively, the BP decoding algorithm possesses intrinsic parallelism and can be efficiently implemented on the targets with highly parallel resources. And, BP decoding provides soft outputs which are necessary for iterative detection and decoding [3]. However, the iterative BP decoder requires large memory and high computation complexity. To solve or alleviate the issues, researchers have proposed some simplified approaches. The XJ-BP decoder [4] utilized different characteristics of the constituent codes in the encoding graph to avoid unnecessary computations within each iteration. In [5], subfactor-graphs convergence reached at earlier stages are considered to reduce the computation complexity. To reduce both memory and computation complexity, the SCAN decoder [6] and its variant RCSC [3] are proposed. However, these simplified approaches, similar with SC decoding, serialize the message updating process and also suffer long decoding latency. On the other hand, to avoid unnecessary decoding iterations and then reduce the computational complexity, researchers have presented early termination criteria (ETC) such as G-matrix and minLLR based methods [7], worst of information bits (WIB) approach [8], frozen bits based scheme [9]. Among these existing ETC, G-matrix based method has the lowest average number of iterations.

Recently, some designers [5] [10–12] concentrated on dedicated hardware BP decoders to offer low-power consumption, low latency and high throughput. However, those hardware solutions have limited flexibility and scalability and require long development cycles. The new generation communication systems such as Software Defined Radio and Virtualized Communication Systems [13] require not only high throughput performance but also high-level flexibility and scalability. To meet those requirements, some researchers [14–16] presented software BP decoders on graphics processing units (GPU). In this work, we present efficient designs of polar BP decoding architectures on GPU to further improve the throughput and latency performance.

The main contribution of this work is as follows: (1) Two effective mapping strategies based on code lengths are proposed to not only reduce the decoding latency but also attain high-level resource utilization; (2) Full shared and global memory efficiency is achieved by adopting 8-bit quantization. In addition, the asynchronous data transfer technique is uti-

lized to solve the unbalanced workloads among the GPU's Streaming Multiprocessors (SMs) and hide the data transfer latency. Compared with [15] at the same BER 10^{-5}, the throughput speedup reaches up to 70.9 times but at the cost of 0.9 dB coding gain loss, which indicates that the proposed decoder provides effective tradeoff between throughput and error performance.

2. POLAR CODES AND BP DECODING ALGORITHM

2.1. Polar codes

Polar codes are constructed on the basis of the channel polarization phenomenon. Channel polarization can be represented by a generator matrix $\mathbf{G} = \mathbf{F}^{\otimes m}$, where $\mathbf{F}^{\otimes m}$ is the m-th Kronecker power of $\mathbf{F} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ and $m = log_2 n$. Besides, the encoding process can be indicated by matrix multiplication $\mathbf{x} = \mathbf{uG}$, where $\mathbf{u} = [u_1, u_2, \ldots, u_n]$ is the n-bit message vector and $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ is the codeword. For an (n, k) polar code, the n positions of $\mathbf{u}$ are grouped into two subsets (A and A^c) according to their corresponding channel reliability. The information set A contains k positions of $\mathbf{u}$ over the k most reliable channels, which are assigned to carry information bits. And, the remaining $(n - k)$ positions constitute the frozen set A^c since the bits at these positions are usually set to 0. The encoding process or matrix multiplication can be also represented by a factor graph, in which m stages are included and each stage includes $n/2$ XOR operations. The encoding graph of an $(8, 4)$ polar code is illustrated in Fig. 1(a).

2.2. BP decoding with G-matrix based ETC

BP decoding is an iterative message-passing algorithm over the encoding factor graph of polar codes shown in Fig. 1(a). For an (n, k) polar code, the factor graph has m stages and each stage includes $n/2$ basic processing elements (PEs). Each node within one PE (see Fig. 1(b)) is associated with two categories of log-likelihood ratio (LLR) messages: left-to-right message $R_{i,j}^{(t)}$ and right-to-left message $L_{i,j}^{(t)}$, where $1 \leq i \leq n$, $1 \leq j \leq m$ and t denotes the iteration number. According to the round-trip schedule, the message $L_{i,j}^{(t)}$ is updated from stage m to 1 and then $R_{i,j}^{(t)}$ is updated from stage 1 to stage m. For the scaled min-sum (SMS) BP algorithm [7], message updating within each PE is as follows:

$$L_{i,j}^{(t)} = \alpha \cdot f(L_{i,j+1}^{(t)}, R_{i+2^{m-j},j}^{(t-1)} + L_{i+2^{m-j},j+1}^{(t)}) \quad (1)$$

$$L_{i+2^{m-j},j}^{(t)} = \alpha \cdot f(L_{i,j+1}^{(t)}, R_{i,j}^{(t-1)}) + L_{i+2^{m-j},j+1}^{(t)} \quad (2)$$

$$R_{i,j+1}^{(t)} = \alpha \cdot f(R_{i,j}^{(t)}, R_{i+2^{m-j},j}^{(t)} + L_{i+2^{m-j},j+1}^{(t)}) \quad (3)$$

$$R_{i+2^{m-j},j+1}^{(t)} = \alpha \cdot f(R_{i,j}^{(t)}, L_{i,j+1}^{(t)}) + R_{i+2^{m-j},j}^{(t)} \quad (4)$$

where α is the scaled parameter (α is 0.9375 [7]) and

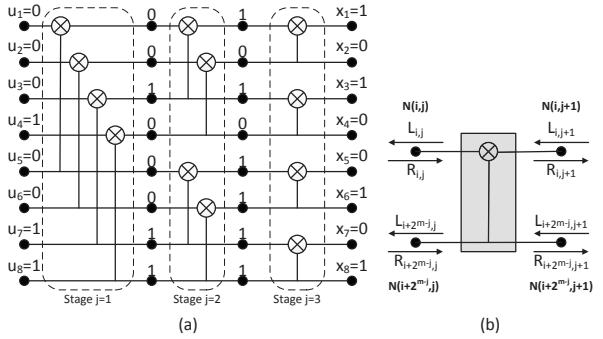

Fig. 1. (a) Encoding factor graph of an $(8, 4)$ polar code ($1 \leq j \leq m$); (b) General processing element. $A = \{4, 6, 7, 8\}$

$f(a, b) = sign(a) \cdot sign(b) \cdot min(|a|, |b|)$. Let $\mathbf{y} = [y_1, y_2, \ldots, y_n]$ be the received message vector over AWGN channel with BPSK modulation. The element $L_{i,m+1}^{(1)}$ is initialized by $\ln \frac{P(y_i|x_i=0)}{P(y_i|x_i=1)}$. $R_{i,1}^{(0)}$ is initialized by $+\infty$ if $i \in A^c$. The remaining elements are initialized by 0.

Let $\hat{\mathbf{u}} = [\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_n]$ and $\hat{\mathbf{x}} = [\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n]$ be the hard decision of $\mathbf{u}$ and $\mathbf{x}$, respectively. After each iteration, $\hat{u}_i$ is updated by $(L_{i,1}^{(t)} + R_{i,1}^{(0)}) > 0?0 : 1$ and $\hat{x}_i$ is updated by $(L_{i,m+1}^{(0)} + R_{i,m+1}^{(t)}) > 0?0 : 1$. If $\hat{\mathbf{u}}\mathbf{G} = \hat{\mathbf{x}}$ or t reaches up to the maximum iteration number (I_{max}), the decoder will output $\hat{\mathbf{u}}$ as the decoding result.

3. OPTIMIZED GPU-BASED BP DECODER FOR POLAR CODES

3.1. Full-parallel BP (FP-BP) decoding architecture for codes $n \leq 2048$

1) On-chip shared memory allocation and accesses. Since the messages $L_{i,j}^{(t)}$ and $R_{i,j}^{(t)}$ between adjacent stages are reused during the decoding procedure, we store those messages on the limited on-chip shared memory to achieve high memory throughput for short code lengths ($n \leq 2048$). For BP decoding, storing the messages $L_{i,j}^{(t)}$ and $R_{i,j}^{(t)}$ of one codeword requires $2n(m + 1)$ memory elements. To save shared memory, 8-bit data unit (*char*) is adopted to represent a message element. Thus, 104KB shared memory are needed for an $n = 4096$ polar code, while the maximum shared memory size is 96KB per SM on a GPU. So, the proposed FP-BP decoder supports the codes with $n \leq 2048$.

Shared memory can work on two modes: 4-byte or 8-byte bandwidth granularity after Fermi architecture. Each SM has 32 banks and each bank is composed of multiple 4-byte or 8-byte memory segments. Avoiding bank conflicts is the main concern of shared memory usage. It will result in bank conflicts that different threads within the same warp access different memory segments of the same bank. Note that there

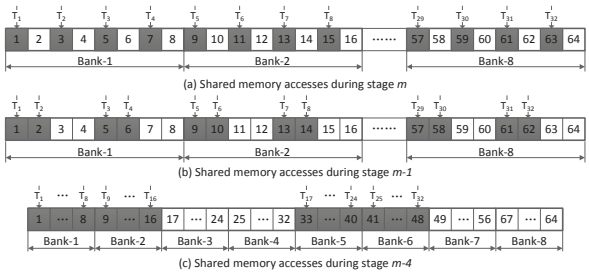

Fig. 2. Shared memory access characteristics for stages $m-4$, $m-1$ and m. $T_1 \sim T_{32}$ represent 32 threads in the same warp. One grey/white block represents 1 byte.

is no bank conflict of the case that several threads within the same warp access different bytes of the same memory segment. We can see that the encoding graph of polar codes in Fig. 1(a) is similar to the butterfly graph of Fast Fourier Transform (FFT). In [17], bank conflicts occur during the last 5 layers of the radix-2 butterfly graph when the banks work on 4-byte granularity and each message element is represented by 4 bytes. Contrary to [17], we use 8-byte mode and each LLR message element is represented by 1 byte (8 bits). The shared memory accesses (stages $m-4$, $m-1$ and m) in this work are illustrated in Fig. 2. Since the data accessed by 32 threads of a warp are 64 bytes, 8 banks are enough to store them and no bank conflict occurs.

2) Mapping strategy of the threads. Considering the parallel characteristics of the BP decoding procedure, we assign one thread block with $n/2$ threads to one polar codeword and the $n/2$ threads are mapped to $n/2$ PEs within the same stage. Thus, the $n/2$ PEs in each stage are simultaneously instantiated during the iterative decoding. The number of active blocks per SM (N_{asm}) is mainly determined by the available shared memory per SM (M_{sm} in KB) and the shared memory size needed by each thread block ($M_b = 2n(m+1)$ in bytes). Here, N_{asm} is given by $\lceil \frac{M_{sm} \times 1024}{M_b} \rceil = \lceil \frac{M_{sm} \times 1024}{2n(m+1)} \rceil$ ($n \leq 2048$) ($\lceil x \rceil$ indicates the largest integer less than or equal to x). And, the maximum number of registers per thread is set to $\lceil \frac{M_{rg} \times 1024 \times 2}{N_{asm} \times n} \rceil$, where M_{rg} is the register size per SM (64KB in general). To reduce the decoding latency and keep the on-chip resource utilization in high level, the total number of codewords processed in parallel (N_b) per asynchronous stream is $N_{asm} \times N_{sm}$, where N_{sm} denotes the number of SMs on the selected GPU.

3.2. Partial-parallel BP (PP-BP) decoding architecture for codes $n > 2048$

1) Global memory accesses. For codes with $n > 2048$, the messages $L_{i,j}^{(t)}$ and $R_{i,j}^{(t)}$ are stored on global memory since the shared memory is not enough. When global memory is adopted, coalesced memory access should be used to achieve full global memory efficiency. During stages 1 to $m-5$, 32 threads of a warp access contiguous global memory addresses and coalesced global accesses can be guaranteed. Note that threads in a warp do not access contiguous global memory from stage $m-4$ to m. Since the cache line size (128 bytes) is twice the area of data addresses (64 bytes) accessed by the 32 threads in a warp, all the message data for one warp can be still cached in Level 1 (L1) cache memory. In addition, for each memory transaction, all data required by two adjacent warps will be cached in the same cache line, which is likely to improve the memory efficiency.

2) Mapping strategy of the threads. For polar codes with length $n > 2048$, similar to the FP-BP decoding architecture, $n/2$ PEs within each stage can be also processed in parallel to reduce the decoding latency. Considering the limited on-chip registers (64 KB per SM), we also assign one block to one codeword but the number of threads per block is set to 1024. Therefore, only 1024 PEs belonging to one stage are simultaneously processed and all PEs from the same stage are partially updated. The maximum number of registers per thread is 32 to fully utilize the on-chip registers, and thus one SM activates two thread blocks. The total number of codewords processed in parallel (N_b) per asynchronous stream is $2 \times N_{sm}$.

3.3. Asynchronous data transfer

Both proposed decoding architectures map one thread block to one codeword. Due to the ETC adoption, the number of decoding iterations varies with the codewords or thread blocks. Thus, thread blocks simultaneously launched on the GPU have different execution time (the blocks with small iteration numbers will finish before the other blocks). The different execution time among the launched thread blocks will result in serious workload unbalance among the SMs and the resource utilization will be lowered accordingly. In this work, we adopt the asynchronous data transfer mode to not only balance the SMs workloads through overlapping the kernels' executions between two adjacent streams but also hide the data transfer latency between host and device.

4. EXPERIMENTAL RESULTS AND DISCUSSIONS

The experimental platform is mainly composed of Intel i7-8700K running at 3.7GHz and NVIDIA GTX TITAN Xp (Pascal architecture, 1405MHz, 30 SMs, 3849 cores, 12GB global memory). The proposed software BP decoder is compiled by CUDA Toolkit 9.2 and Visual Studio 2013 on windows 7 x64 system. Fig. 3 shows the BER performance and the average number of iterations of the proposed GPU-based decoder with adoption of the G-matrix based ETC and round-trip schedule ($I_{max} = 40$).

For the FP-BP decoder, to fully utilize on-chip registers and shared memory according to Section 3.1, the maximum

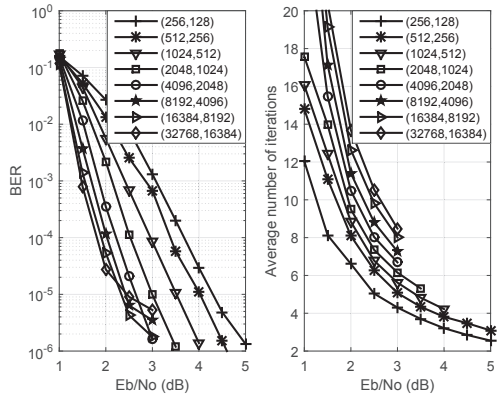

Fig. 3. BER performance and average number of iterations of the proposed decoders over different Eb/No values.

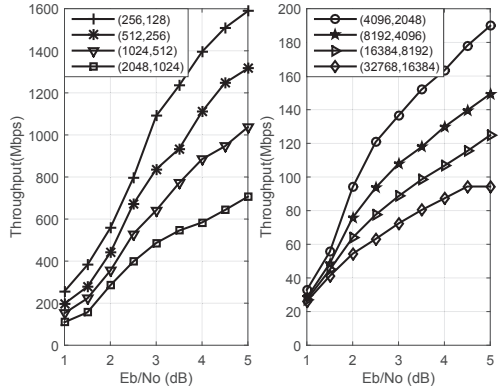

Fig. 4. Throughput performance of the proposed decoders over different Eb/No values.

registers per thread are set to 24, 28, 32, 32 and the thread blocks per stream are 630, 270, 120, and 60 for code lengths 256, 512, 1024, and 2048. For the PP-BP decoder, the thread blocks per stream are 60 and the maximum registers per thread are 32. The number of asynchronous streams is set to 5 for all codes. The decoding process contains the data transfer from host to device, iterative decoding, G matrix-based ETC procedure, and the data transfer from device to host. Fig. 4 and Fig. 5 summarize the throughput and latency of the proposed decoders over different Eb/No values, respectively. For codes with length $n \leq 1024$, the FP-BP decoder achieves more than 1 Gbps throughput and less than 1 *ms* latency at 5 dB.

Table 1 gives the decoding throughput comparison with [14] at 25 iterations and [15] at 50 iterations. For fair comparison, we adopt the metric named TDNC (Throughput under Normalized Decoding Cost, Mbps per SM per MHz) presented in [18]. The throughput comparison with [14] is still conducted in this work to comprehensively evaluate the proposed

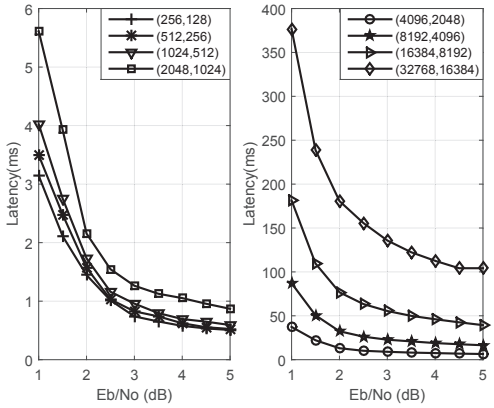

Fig. 5. Latency performance of the proposed decoders over different Eb/No values.

Table 1. Throughput comparison with related works

N	$Eb/N0$ (dB)	Related works		Ours	Speed up
		Ref.	TNDC	TNDC	
256	4.0	[14]	1.837	38.0	20.7
512			0.911	31.2	34.2
1024			0.371	23.8	64.3
2048			0.129	16.9	131
4096	2.0	[15]	0.23	2.43	10.7
	3.0		1.045	3.44	3.29
	4.0		1.545	4.010	2.59
	5.0		1.545	4.415	2.86

decoder, even though we have difficulty reproducing the error performance shown in [14]. Under the same BER performance such as 10^{-5}, the proposed BP decoder achieves about 70.9 times throughput speedup than [15] but suffers about 0.9 dB coding gain loss due to the combination of SCL and BP decoding in [15].

5. CONCLUSION

This paper presents an FP-BP decoding architecture for codes $n \leq 2048$ and a PP-BP decoding architecture for codes $n > 2048$ with efficient memory allocation and mapping strategies. To achieve high resource utilization, 8-bit quantization and the asynchronous data transfer mode are adopted. Compared with the related works, much higher throughput can be achieved, especially for short codes. For codes with length $n \leq 1024$, the proposed TITAN Xp decoder achieves above 1 Gbps throughput and less than 1 ms latency by using the G-matrix based ETC. The presented GPU-based decoder can be used as a flexible channel decoding module in the new generation communication systems.

6. REFERENCES

[1] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051-3073, Jul. 2009.

[2] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2213-2226, May 2015.

[3] J. Lin, C. Xiong, and Z. Yan, "Reduced complexity belief propagation decoders for polar codes," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, Oct. 2015.

[4] J. Xu, T. Che, and G. Choi, "XJ-BP: Express journey belief propagation decoding for polar codes," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, Dec. 2015.

[5] S. M. Abbas, Y. Fan, J. Chen and C. Tsui, "High-throughput and energy-efficient belief propagation polar code decoder," *IEEE Transaction On Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 3, pp. 1098-1111, Mar. 2017.

[6] U. U. Fayyaz and J. R. Barry, "Low-complexity soft-output decoding of polar codes," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 958-966, May 2014.

[7] B. Yuan and K. K. Parhi, "Early stopping criteria for energy-efficient low-latency belief-propagation polar code decoders," *IEEE Trans. Signal Process.*, vol. 62, no. 24, pp. 6496-6506, Dec. 2014.

[8] C. Simsek and K. Turk, "Simplified early stopping criterion for belief-propagation polar code decoders," *IEEE Comm. Lett.*, vol. 20, no. 8, pp. 1515-1518, Aug. 2016.

[9] Q. Zhang, A. Liu and X. Tong, "Early stopping criterion for belief propagation polar decoder based on frozen bits," *Electronics Letters*, vol. 53, no. 24, pp. 1576-1578, Nov. 2017.

[10] B. Yuan and K. K. Parhi, "Architecture optimizations for BP polar decoders," in *Proc. IEEE Int. Conf. Acoust. Speech, Signal Process. (ICASSP)*, May 2013, pp. 2654-2658.

[11] Y. S. Park, Y. Tao, S. Sun, and Z. Zhang, "A 4.68Gb/s belief propagation polar decoder with bit-splitting register file," in *Proc. IEEE Int. Symp. VLSI Circuits Dig. Tech. Papers*, Jun. 2014, pp. 1-2.

[12] B. Yuan and K. K. Parhi, "Architectures for polar BP decoders using folding," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jun. 2014, pp. 205-208.

[13] S. Sun, M. Kadoch, L. Gong, and B. Rong, "Integrating network function virtualization with SDR and SDN for 4G/5G networks," *IEEE Network*, vol. 29, no. 3, pp. 54-59, Jun. 2015.

[14] B. K. Reddy L. and N. Chandrachoodan, "A GPU implementation of belief propagation decoder for polar codes," in *Proc. Asilomar Conf. on Signals, Systems, and Computers*, Nov. 2012, pp. 1272-1276.

[15] S. Cammerer, B. Leible, M. Stahl, J. Hoydis and S. T. Brink, "Combining belief propagation and successive cancellation list decoding of polar codes on a GPU platform," in *Proc. IEEE Int. Conf. Acoust. Speech, Signal Process. (ICASSP)*, pp. 3664-3668, Mar. 2017.

[16] J. Yin, Q. Huang, L. Li, A. Gao, W. Chen and Z. Han, "High throughput parallel encoding and decoding architecture for polar codes," in *Proc. IEEE/CIC International Conference on Communications in China (ICCC)*, Oct. 2017.

[17] J. Andrade, G. Falcao and V. Silva, "Optimized fast walsh-hadamard transform on GPUs for non-binary LDPC decoding," *Parallel Computing*, vol. 40, no. 9, pp. 449-453, Oct. 2014.

[18] Y. Ying, K. You, L. Zhou, H. Quan, M, Jing, Z. Yu and X. Zeng, "A pure software LDPC decoder on a multi-core processor platform with reduced inter-processor communication cost," in *Proc. IEEE International Symposium on Circuits and Systems*, May 2012, pp. 2609-2612.