GPU-Based Gigabit LDPC Decoder

Selcuk Keskin and Taskin Kocak

***Abstract*— In this letter, we present design and implementation of a parallel software low-density parity-check (LDPC) decoding algorithm on graphics processing units (GPUs). As a solution for the LDPC decoder, dedicated application-specific integrated circuit or field programmable gate array implementations is proposed in recent years in order to support high throughput despite their long deployment cycle, high design, and especially fixed functionalities. On the other hand, the implementations on GPU as a software solution provide flexible, scalable, and less expensive solutions in a shorter deployment cycle. We present some GPU-based optimizations for a major LDPC decoder algorithm to obtain high throughput in digital communication systems. Experimental results demonstrate that the proposed LDPC decoder achieves more than 1.27 Gb/s peak throughput on a single GPU.**

***Index Terms*— High throughput, decoding, MSA, LDPC codes, CUDA, GPU, concatenated decoders.**

I. Introduction

LDPC codes were originally proposed by Robert Gallager in 1962 and re-discovered by Mackay and Neal [1] in 1996. They have been used in digital communication systems, such as WiMAX (802.16e), WiFi (802.11n), DVB-S2, etc. Tanner introduced an effective graphical representation for LDPC Tanner codes to describe the decoding algorithm [2]. With the advent of technology, the interests in LDPC codes have been dramatically increased because their excellent error-correcting performance is much closer to the Shannon limit [3]. LDPC can be decoded with iterative soft-decision decoding algorithms called message-passing algorithms [4]. Error correcting has started to become a challenge because of complex computations. The belief propagation algorithm can be simplified using the min-sum approximation, which greatly reduces the implementation complexity, but incurs a degradation in decoding performance.

A GPU provides a parallel architecture, which combines raw computation power with programmability. GPUs have started to offer high performance general purpose processing by executing thousands of threads simultaneously. A GPU provides extremely high computational throughput by employing many cores working on a large set of data in parallel. In this work, we examine whether the massively parallel computation approach of GPU can be used for the decoding process.

II. LDPC Decoding Algorithm

Low-density parity-check (LDPC) codes are a class of linear block codes. The name comes from the characteristic of their parity-check matrix which contains only a few *1*'s in comparison to the number of *0*'s. Their main advantage is that they provide performance which is very close to the capacity for a lot of different channels and linear time complex algorithms for decoding. LDPC decoding requires the propagation of messages between connected nodes, as indicated by the Tanner graph. The two types of nodes in a Tanner graph are called bit nodes (BN) and check nodes (CN). Tanner graphs have proved to be an efficient algorithm for inference calculation, and it is used in numerous applications. LDPCs are linear (n, k) block codes defined by sparse binary parity-check H matrices of dimension $m \times n$, with $m = n-k$ and $rate = k/n$.

(1944,972) Code rate 1/2

57				50		11		50		79		1	0										
3		28		0				55	7				0	0									
30				24	37			56	14					0	0								
62	53			53			3	35							0	0							
40			20	66			22	28								0	0						
0				8		42		50			8						0	0					
69	79	79				56		52				0						0	0				
65				38	57			72		27										0	0		
64				14	52			30			32											0	0
	45		70	0				77	9													0	0
2	56		57	35						12													
24		61		60			27	51			16	1											0

Fig. 1. Permutation indices of parity check H matrix for FEC rate-1/2 LDPC code in 802.11n standard.

The parity check H matrix of IEEE802.11n specification for the FEC rate 1/2 with 6966 connected nodes which is represented by permutation matrices can be seen in Fig. 1. Each value is the cyclic-permutation matrix p^i that is obtained from the 81×81 identity matrix by cyclically shifting the columns to the left by i elements. The matrix p^0 is the 81×81 identity matrix.

There are two major computation algorithms for LDPC decoding in literature, the first one is Sum-Product Algorithm (SPA) which is a belief propagation and its simplified approximation, Min-Sum Algorithm (MSA) [4]. Both of them are iterative methods and they use hard coding mechanism before ending. In this letter, the MSA is designed and implemented for parallel execution on massive number of cores in the GPUs. The MSA is introduced to decrease the computational complexity of the check nodes processing in the belief propagation algorithm. MSA is mainly described by two different horizontal and vertical intensive processing blocks. The input data are Log-Likelihood Ratios (LLRs). Horizontal processing updates the messages from CN_m to BN_n with,

$$Lr^i_{mn} = \min_{n' \in N(m) \backslash n} \left| Lq^{i-1}_{n'm} \right| \prod_{n' \in N(m) \backslash n} \operatorname{sign}(Lq^{i-1}_{n'm}) \qquad (1)$$

where Lr_{mn} are LLRs at check nodes and Lq_{nm} are LLRs at bit nodes. Similarly, vertical processing computes the messages sent from BN_n to CN_m with:

$$Lq^i_{nm} = Lp_n + \sum_{m' \in M(n) \backslash m} Lr^i_{m'n} \qquad (2)$$

Manuscript received April 6, 2017; accepted May 10, 2017. Date of publication May 11, 2017; date of current version August 10, 2017. A part of this work is financially supported by KDDI R&D Laboratories Inc., Japan. We also acknowledge the support of NVIDIA Corporation with the donation of the GPU used for this research. The associate editor coordinating the review of this paper and approving it for publication was J. Li. *(Corresponding author: Selcuk Keskin.)*

The authors are with the Department of Computer Engineering, Bahcesehir University, 34353 Istanbul, Turkey (e-mail: selcuk.keskin@eng.bau.edu.tr; taskin.kocak@eng.bau.edu.tr).

Digital Object Identifier 10.1109/LCOMM.2017.2704113

where Lp_n are initial LLRs. Finally, at the end of the algorithm when maximum number of iterations is reached, hard decision is performed with the below equation:

$$Q_n = Lp_n + \sum_{m' \in M(n)} Lr_{m'n} \tag{3}$$

where Q_n are the a posteriori pseudo-probabilities.

III. GPU Based LDPC Decoding

The parallelization of algorithms is a challenging process. The number of threads and the amount of memories must be decided carefully, also the limited number of resources should be utilized fully. The design must be realized with a minimum latency because of frequent communication between the threads. In addition, attention should be paid to the synchronization to get correct results. Compute Unified Device Architecture (CUDA) is a parallel computing platform and application programming interface model created by NVIDIA [5]. CUDA allows software developers to use a CUDA-enabled GPU for general purpose processing.

Since GPU kernels need more initialization time than CPU functions, multiple concurrent codewords must be transferred to the global memory of GPU instead of calculating a single codeword. Other GPU related optimizations for proposed LDPC decoder are described in subsections.

A. Using Constant Look-Up Tables for the H Matrix Data

The Compressed Column Storage (CCS) format puts the subsequent non-zeros of the matrix columns in contiguous memory locations [6]. By using CCS format, H matrix can be represented with only 0.67% of the elements for FEC rate 1/2 of IEE 802.11n standard. All 1s of H matrix can be represented by using four different vectors; rowDegree, rowIndex, columnDegree and columnIndex.

These vectors are defined in constant memory of GPU to obtain better memory bandwidth because of temporal locality. Since constant memory is a read-only cache, access time is less than the global memory. During GPU computation, each thread reads the indices from the constant memory.

B. Optimization of Global Memory Input Access

Taking into account that many accesses will be requested, processing on global memory will take too much time. Frequently used data should be moved to a faster memory level, i.e. shared memory. As a first step of the algorithm, all inputs are transferred to shared memory by parallel threads with coalesced structure. To perform the first row processing, some threads must need to read from the same locations which are indexed by an array; namely, $Hrow$ which is created with column based rank of 1s in H matrix as shown in Fig 2. Reading from the same location on the shared memory is not as problematic as the GPU global memory.

C. Data Access Method

HCN array holds the column-wise ranks of each 1 position in H matrix. In the column process, inputs are read from contiguous memory locations as well as outputs are written

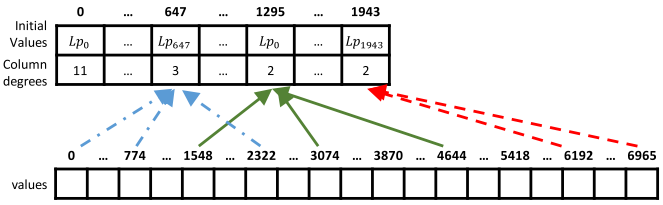

Fig. 2. Creating values pattern by using initial inputs.

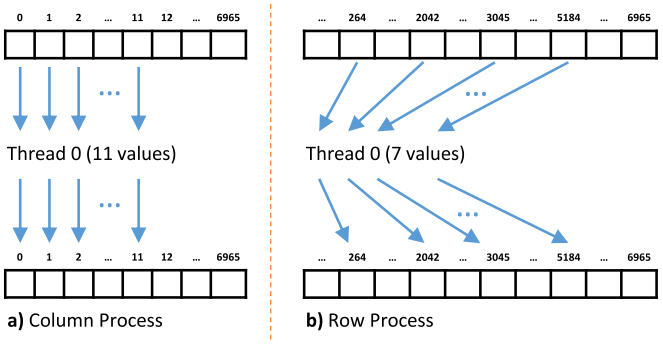

Fig. 3. Memory access pattern.

to contiguous memory locations. Each thread updates up to 11 values during column processing. During row processing, the inputs are being read and written with using HCN array. The memory access pattern is illustrated for the first thread in Fig. 3. As the last step of decoding, hard decoded values are stored into global memory in a memory coalesced fashion.

Algorithm 1 Life Cycle of a Thread During Decoding of a Single Codeword

Require: input ▷ Log-Likelihood Ratios
Ensure: output ▷ Hard decoded values $\in \{0, 1\}$
$xIndex = N \times bIdx.x + tIdx.x$ ▷ (bIdx: blockID)
$yIndex = M \times bIdx.x + tIdx.x$ ▷ (tIdx: threadID)
Lp_n values are initialized with $input$ according to $xIndex$
Load SMEM and perform the first row processing
for $i = 1 \rightarrow I_{max}$ **do**
  // Perform column processing with coalescing read & write
  Read Lq_{nm} from $SMEM[colIndex[tIdx.x]]$
  Update Lr_{mn} with using Eq. (1)
  Write Lr_{mn} to $SMEM[colIndex[tIdx.x]]$
  // Perform row processing with HCN array
  Read Lr_{mn} from $SMEM[HCN[rowIndex[tIdx.x]]]$
  Update Lq_{nm} with using Eq. (2)
  Write Lq_{nm} to $SMEM[HCN[rowIndex[tIdx.x]]]$
end for
// Perform hard decoding
Calculate Q_n by using Eq. (3)
If $Q_n > 0$ **then** $output[yIndex] = 1$
else $output[yIndex] = 0$

General outlines of the algorithm on GPU are illustrated in Algorithm 1. Also, Figure 4 shows the flow of MSA decoding algorithm on GPU for FEC rate 1/2 of IEEE802.11n.

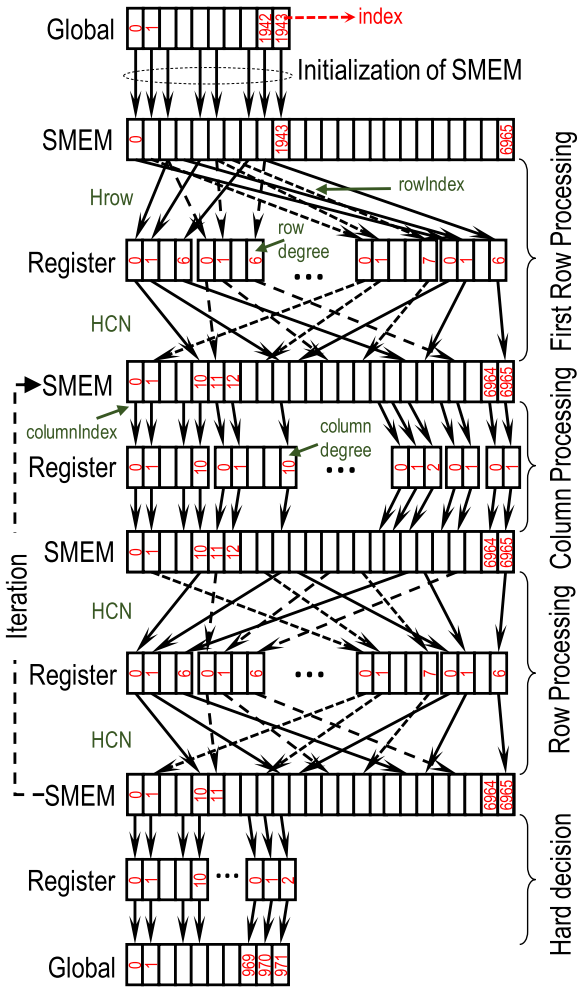

Fig. 4. Algorithm representation on GPU architecture (Note that, the numbers in boxes show the indices of current item of array, not the values).

IV. Experimental Results

In this letter, the decoding algorithm is implemented on Pascal architecture based NVIDIA TITAN X which has 8 billion transistors, 3584 CUDA cores and 12GB of GDDR5 memory. The NVIDIA GPU architectures consist of multiple stream multiprocessors(SM) which consist of pipelined cores and instruction dispatch units.

To evaluate bit error rate (BER) performance of the GPU based MSA decoder, a simulator platform with the Matlab original code is based for comparison. Figure 5 illustrates simulation results for 16-QAM under the AWGN channel. It shows that GPU based MSA decoder obtains nearly same performance with Matlab.

Early termination is not used in our GPU based MSA implementation to make fair comparison. Thus, the algorithm continues until the the number of maximum iterations. To see the real performance of proposed GPU based MSA implementation, we also examined that how many iterations actually we need to get best BER results. MSA reaches to the best result earlier than Matlab decoder in each Eb/No value as shown in Fig. 6. The results show that 5 iterations are actually enough to decode at low Eb/No values.

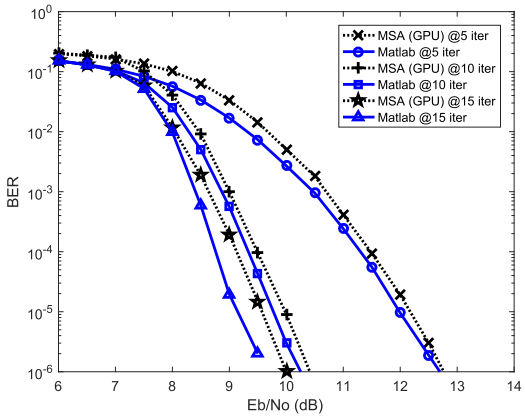

Fig. 5. BER comparison of algorithms for FEC rate = 1/2.

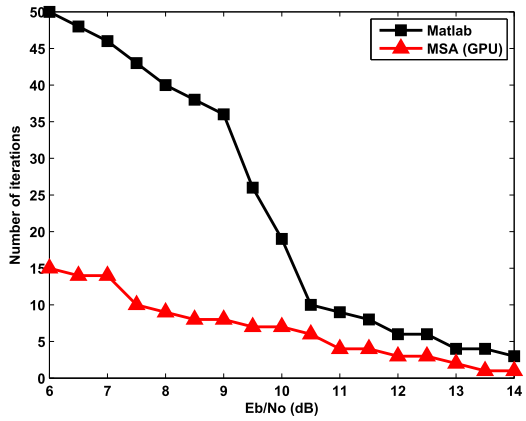

Fig. 6. Performance of LDPC Decoder algorithm.

TABLE I
THROUGHPUT AND LATENCY PERFORMANCE AT DIFFERENT NUMBER OF ITERATIONS {5,10,15}

FEC	Latency (μs)			Throughput (Mbps)		
	5	10	15	5	10	15
1/2	1.52	2.13	2.35	1276	913	826
2/3	1.46	2.03	2.24	1335	957	866
3/4	1.70	2.37	2.62	1145	821	743
5/6	2.12	2.95	3.27	917	658	595

The kernel is processed for multiple concurrent codewords and the average time is recorded. The data transfer time between host and device memory is limited by PCI bus bandwidth. It is independent to the performance of code running time on GPU, so the transfer time is excluded from our experimental results. Thanks to the flexibility of GPU based software solution, we easily implemented all FEC rates of IEEE 802.11n specifications. Table I shows the decoding time and the throughput of these codes at different iterations.

Table II shows the comparison of the results of this work against the recently reported GPU-based LDPC decoder implementations. The results are given for 10 iterations except Li *et al.* [12] uses 5 iterations. Results reported in Table II clearly demonstrate that the throughput achieved by our decoder is higher than the-state-of-the-art. Because each work

TABLE II
PERFORMANCE COMPARISON OF VARIOUS GPU-BASED LDPC DECODERS

	[7]	[8]	[9]	[10]		[11]	[12]	This Work
Block Code	(2048,1723)	(7168,3072)	(8000,4000)	(1944,972)	(2304,1152)	(4000,2000)	(2304,1152)	(1944,972)
# of edges	12288	19456	24000	6966	7296	12000	7296	6966
GPU	GTX480	GTX580	Tesla C2050	GTX TITAN		GTX660Ti	GTX580	TITAN X
Throughput (Mbps)	146.6	121.58	209	236.70	304.16	515	712	913
Throughput per core	0.31	0.21	0.31	0.09	0.11	0.38	1.39	0.25
Latency (μs)	13.97	58.96	38.28	8.21	7.29	7.77	3.24	2.13
Latency per edge (ns)	1.13	3.03	0.66	1.17	1.03	0.64	0.44	0.31

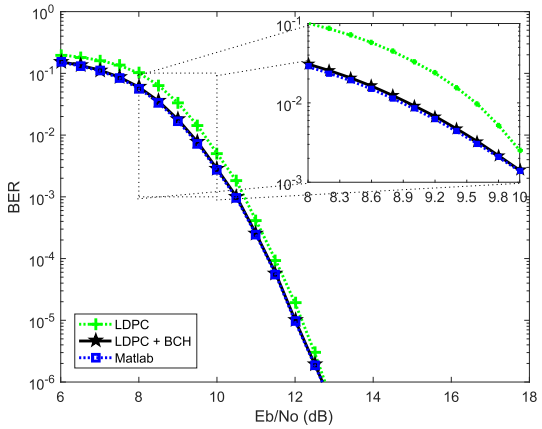

Fig. 7. BER performance of the coding systems concatenating LDPC code with BCH code.

uses different number of codewords, latency per codeword is shown in the table to make fair comparison. Also, the resource consumption for different kinds of LDPC codes is very different. Thus, Table II shows also decoding time per edge that the is number of ones in parity check matrix.

V. DISCUSSION

Our proposed GPU based LDPC decoder achieve gigabit level throughput. Hence, we have some extra decoding time as such a second decoder can be performed to improve BER performance. LDPC can be concatenated with Bose-Chaudhuri-Hochquenghem (BCH) codes to meet higher data rates for any FEC rate with closer Shannon limit performance due to the increasing demand for high-speed communication. Concatenated codes have been widely used in variety of communication systems such as space communication links, magnetic recording systems, and wireless systems. Thanks to scalable and flexible structure of GPU based algorithms, the proposed GPU based LDPC decoder can be used as inner code to achieve excellent error-correcting performance.

The data is encoded by an outer BCH code encoder and then each segment is encoded by an inner LDPC code encoder. In the receiver part, all codewords are first decoded by the LDPC decoder and then the BCH outer decoder is executed to reduce the errors that caused by LDPC decoding. The BER performance of concatenated decoder is illustrated in Fig. 7. Proposed LDPC and Matlab with 5 iterations algorithms are selected to compare. The BER performance of concatenated system is very close to Matlab's performance. However, we need to budge from throughput performance because the decoding time increases from nanoseconds to microseconds. The proposed GPU based concatenated decoder achieves over 300 Mbps with 5 iterations.

VI. CONCLUSIONS

This letter presents the design, methodology and implementation of a major LDPC decoder on GPU architecture. GPUs are resourceful and efficient for accelerating DSP-based algorithms; however, the challenge is to map the algorithms to the computational resources in GPUs. To achieve high decoding throughput, several techniques including the algorithmic optimizations, efficient data structures and memory access optimizations are used as enhancements on the basic form of the algorithm. The experimental results show that the proposed GPU based LDPC MSA decoder achieves up to 1276 Mbps peak throughput with a competitive BER performance which outperforms the previously related works. Also, the GPU based LDPC code can be used in concatenated decoding systems to meet the error protection and high-speed communication requirement of applications.

REFERENCES

[1] D. Mackay and R. Neal, "Near Shannon limit performance of low density parity check codes," *IEEE Electron. Lett.*, vol. 32, no. 18, pp. 1645–1646, Mar. 1996.
[2] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.
[3] X.-Y. Shih, C.-Z. Zhan, C.-H. Lin, and A.-Y. Wu, "An 8.29 mm^2 52 mW multi-mode LDPC decoder design for mobile WiMAX system in 0.13 μm CMOS process," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, pp. 672–683, Mar. 2008.
[4] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Characterization of belief propagation and its generalizations," Mitsubishi Elect. Res. Lab. Inc., Cambridge, MA, USA, Tech. Rep. TR2001-15, 2001.
[5] (2013). *The CUDA Programming Guide*. [Online]. Available: https://developer.nvidia.com/category/zone/cuda-zone
[6] I. S. Duff, R. G. Grimes, and J. G. Lewis, "Sparse matrix test problems," *ACM Trans. Math. Softw.*, vol. 15, no. 1, pp. 1–14, 1989.
[7] S. Kang and L. Moon, "Parallel LDPC decoder implementation on GPU based on unbalanced memory coalescing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2012, pp. 3692–3697.
[8] Y. Hou, R. Liu, H. Peng, and L. Zhao, "High throughput pipeline decoder for LDPC convolutional codes on GPU," *IEEE Commun. Lett.*, vol. 19, no. 12, pp. 2066–2069, Dec. 2015.
[9] G. Falcao, V. Silva, L. Sousa, and J. Andrade, "Portable LDPC decoding on multicores using OpenCL," *IEEE Signal Process. Mag.*, vol. 29, no. 4, pp. 81–109, Apr. 2012.
[10] G. Wang, M. Wu, B. Yin, and J. R. Cavallaro, "High throughput low latency LDPC decoding on GPU for SDR systems," in *Proc. IEEE Global Conf. Signal Inf. Process. (GlobalSIP)*, Dec. 2013, pp. 1258–1261.
[11] Y. Lin and W. Niu, "High throughput LDPC decoder on GPU," *IEEE Commun. Lett.*, vol. 18, no. 2, pp. 344–347, Feb. 2014.
[12] 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 Comput. Pract. Exper.*, vol. 27, no. 1, pp. 29–46, 2013.