2013000436

Engineering Village

经检索“**Engineering Village 2**”，以下论文被《**Ei Compendex**》收录(含“**Ei controlled terms**”的论文)。(检索时间**2013**年**9**月**2**日)。

姜斌

<RECORD 1>
Accession number:20133216593079
Title:Efficient simulation of QC LDPC decoding on GPU platform by CUDA
Authors:Jiang, Bin (1); Bao, Jianrong (2); Xu, Xiaorong (2)
Author affiliation:(1) School of Telecommunication Engineering, Hangzhou Dianzi University Hangzhou, Zhejiang, 310018, China; (2) Zhejiang Provincial Key Laboratory of Information Network Technology, Zhejiang University, Hangzhou, Zhejiang, 310027, China
Source title:2012 International Conference on Wireless Communications and Signal Processing, WCSP 2012
Abbreviated source title:Int. Conf. Wirel. Commun. Signal Process., WCSP
Monograph title:2012 International Conference on Wireless Communications and Signal Processing, WCSP 2012
Issue date:2012
Publication year:2012
Article number:6542885
Language:English
ISBN-13:9781467358293
Document type:Conference article (CA)
Conference name:2012 International Conference on Wireless Communications and Signal Processing, WCSP 2012
Conference date:October 25, 2012 - October 27, 2012
Conference location:Huangshan, China
Conference code:98103
Sponsor:IEEE Communications Society Nanjing Chapter; IEEE Signal Processing Society Nanjing Chapter; University of Science and Technology of China; Huawei Technologies Co., Ltd.
Publisher:IEEE Computer Society, 2001 L Street N.W., Suite 700, Washington, DC 20036-4928, United States
Main heading:Decoding
Controlled terms:Computer simulation - Parallel architectures - Signal processing - Wireless telecommunication systems
Uncontrolled terms:CUDA - GPU - LDPC codes - Parallel Computation - QC
Classification code:716 Telecommunication; Radar, Radio and Television - 716.1 Information Theory and Signal Processing - 717 Optical Communication - 722 Computer Systems and Equipment - 723 Computer Software, Data Handling and Applications - 723.5 Computer Applications
DOI:10.1109/WCSP.2012.6542885
Database:Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.

注：
以上检索结果均得到被检索人的确认。

《Engineering Index》检索结果
杭州电子科技大学图书馆信息咨询部
检索人(签章): [illegible]
2013年9月2日

2012 International Conference on Wireless Communications and Signal Processing

October 25-27, 2012, Huangshan, China

WCSP 2012
October 25-27,2012,Huangshan,china
IEEE
IEEE COMMUNICATIONS SOCIETY

Papers by Session

CTS-02: LDPC codes

Chair: Xiaofu Wu (Nanjing University of Posts & Telecommunications, P.R. China)

Click on a title for a paper.

Efficient Simulation of QC LDPC Decoding on GPU Platform by CUDA

Bin Jiang
School of Telecommunication Engineering
Hangzhou Dianzi University
Hangzhou, Zhejiang, P.R. China, 310018
jiangbin@hdu.edu.cn

Jianrong Bao , Xiaorong Xu
Zhejiang Provincial Key Laboratory of
Information Network Technology, Zhejiang University,
Hangzhou, Zhejiang, P.R. China, 310027
{baojr, xuxr}@hdu.edu.cn

***Abstract*—An efficient parallel simulation scheme of quasi-cyclic (QC) low-density parity-check (LDPC) decoding is proposed to improve the simulation efficiency greatly. It employs multi-threads with the multi-processors of a graphic processing unit (GPU) to perform the simulation of LDPC decoding in parallel. Other than full hardware based LDPC decoding, it obtains good features of low cost, easy programming complexity by using the compute unified device architecture (CUDA) techniques. The CUDA also provides parallel computing by the GPU with efficient multi-thread computation and very high memory bandwidth. Based on the proposed scheme, all bit nodes or check nodes can be updated in an LDPC decoding iteration simultaneously. Therefore, it provides an efficient and fast approach of QC LDPC decoding.**

Keywords- LDPC codes; QC; parallel computation; GPU; CUDA

I. INTRODUCTION

LDPC codes are a class of Shannon capacity approaching channel codes developed recently [1, 2]. They can be decoded efficiently by the belief propagation (BP) algorithm or equivalent sum-product (SP) algorithm [2, 3]. Moreover, they have been implemented with hardware in parallel [4]. QC LDPC codes are one category of LDPC codes where their parity check matrixes have QC structure. The QC structure not only makes the encoding much easier but also reduces the decoding complexity for their easy addressing, i.e., finding the "1" in the matrix [5]. However, the simulations of their decoding cost much time in a personal computers (PCs) when the requirement of the bit-error-rate (BER) performance is very high such as 10^{-7}. In addition, the energy costs are huge since it may simulate just for a single code performance.

Other than PC computation platforms, there are some other efficient simulation platforms for LDPC decoding as the field programmable gate array (FPGA) simulation platforms [5], etc. The FPGA platform greatly accelerates the simulation speed. But the cost of the FPGA platform is expensive and it is not flexible enough to adopt different LDPC decoding simultaneously. In addition, it has quantization effect due to limited resources of physical and logical units. Recently, GPU with CUDA techniques [6, 7] are introduced in the communication calculations as LDPC decoding, biomedical analyses and so on. [8, 9] introduce the LDPC decoding simulations in a GPU with CUDA techniques. But they just introduce the general LDPC decoding in the GPUs. For LDPC codes with special QC structure, we introduce a more efficient algorithm proper for the GPU platform by adopting QC structure to fit the multi-threads GPU processing and simplify the calculations.

This paper is organized as follows. In section II, the principles of the QC LDPC decoding is introduced. In Section III, the parallel simulation of QC LDPC decoding on GPU using CUDA is proposed to implement efficient simulation platform. Then the computation complexity is analyzed and compared with the scheme of simulation on central processing unit (CPU) in section IV. Finally, the conclusion is drawn in section V.

II. LDPC DECODING WITH LOGARITHM BP ALGORITHM

BP algorithm is efficient for LDPC decoding [1, 2]. Its logarithm version (log-BP) can be described as follows.

Some notations: The set of the *j*-th variable node connected to the *i*-th check node is $K(j)=\{i:\mathbf{H}_{i,j}=1\}$ and $\mathbf{H}_{i,j}$ is the element of the check matrix $\mathbf{H}$ of the code with index (i, j). The set of the *i*-th check node connected to the *j*-th variable node is $M(i)=\{j:\mathbf{H}_{j,i}=1\}$. $L(c_i)$ is the initial logarithm likelihood ratio (*LLR*) of the codeword c_i. x_i and y_i are the *i*-th decided and received signal respectively. r_{ji} and q_{ij} are the messages between the variable node *j* and the check node *i*. Their *LLRs* are $L(r_{ji})=\ln[r_{ji}(0)/r_{ji}(1)]$ and $L(q_{ij})=\ln[q_{ij}(0)/q_{ij}(1)]$ where $r_{ji}(u)$ and $q_{ij}(u)$ are the messages of r_{ji} and q_{ij} decided as 0 or 1. Q_i is the *LLR* of the *i*-th variable node of the codeword c_i. N is the code length. $\mathbf{C}=[c_1,\ldots,c_N]^T$ is the codeword vector.

At an AWGN with variance σ^2, the LDPC decoding can be expressed as below:

Step (1) Initialization

$$L(c_i)=\ln[\frac{p(x_i=1\mid y_i)}{p(x_i=-1\mid y_i)}]=2y_i/\sigma^2 \tag{1}$$

$$L(q_{ij})=L(r_{ji})=0 \tag{2}$$

Step (2) Decoding process (iterations between the variable and check nodes)

Step (2.1) Update the variable nodes and perform the final decoding judgment.

$$L(q_{ij})=L(c_i)+\sum_{j'\in M(i)\setminus j}L(r_{j'i}) \tag{3}$$

$$L(Q_i)=L(c_i)+\sum_{j\in M(i)}L(r_{ji}) \tag{4}$$

$$\hat{c}_i=\begin{cases}0, & L(Q_i)\geq 0\\ 1, & L(Q_i)<0\end{cases} \tag{5}$$

In this procedure, if $\boldsymbol{H}\cdot\hat{\boldsymbol{C}}-\boldsymbol{0}$ (Mod 2), $\hat{C}$ is the final result. And if $\hat{C}$ is the decoding result or the iteration exceeds the maximum iteration times, it is finished. Otherwise, go to step (2.2).

Step (2.2) Update the check nodes and then go to step (2.1) to carry over the iterations.

$$L(r_{ji}) = 2\tanh^{1}\{\prod_{i'\in K(j)\backslash i} \tanh[L(q_{i'j})/2]\} \qquad (6)$$

And (6) can be calculated with much lower complexity by some numerical processing. Firstly, some notations are defined as follows.

$$L(q_{i'j}) = \alpha_{i'j}\beta_{i'j} \qquad (7)$$

$$\alpha_{i'j} = sign[L(q_{i'j})] \qquad (8)$$

$$\beta_{i'j} = |L(q_{i'j})| \qquad (9)$$

$$\phi(x) = \log\tanh(x/2) = \log\frac{e^x+1}{e^x-1} \qquad (10)$$

Then (6) can be calculated by (11).

$$L(r_{ji}) = (\prod_{i'\in K(j)\backslash i}\alpha_{i'j})\phi[\sum_{i'\in K(j)\backslash i}\phi(\beta_{i'j})] \qquad (11)$$

The LDPC decoding algorithm can be processed mainly by additions and computations of $\phi(x)$ which can be simply solved in a function value table. And the QC structure of the LDPC code makes the check matrix more regular. So the whole decoding can be efficiently performed with parallel structure with easy addressing of 1 in the matrix. Therefore, it can be computed in the multi-core processors in parallel to accelerate the intensive computations.

III. QC LDPC DECODING BY GPUS USING CUDA

A. Blief Introductions of the CUDA techniques

CUDA is a parallel computing architecture based on the GPU developed by the well-known graphic corporation, NVIDIA [6]. And it can be served as a general computing device by direct accessing to the low-level hardware based graphics applicable programmable interfaces (APIs) [8]. The GPU has more co-processors than the CPU and it is capable of running multi-thread programs simultaneously for high computation throughputs. With the CUDA techniques, the GPU can execute numerical computations by employing a great number of threads from the multi-core co-processors in parallel. The GPU can operate as a coprocessor to the CPU and it mainly process data-parallel, compute-intensive portions of the calculations.

In the GPU computation, the threads can cooperate by sharing data through the fast shared memory and executing programs in parallel. It can copy data from the host (CPU) to the device (GPU) and vice versa through the low-level API calls that utilizes the device's direct memory access (DMA) with high performance [6]. Original GPU programming is difficult since it uses special hardware based instructions to programming. But NVIDIA has developed a new easily programmable CUDA technique to overcome this deficiency by the CUDA's parallel programming model with a C-like standard programming language. Therefore, the GPU with CUDA technique is especially well-suited for solving such intensive calculations where the same program is executed on many threads efficiently.

The batch of the multi-threads that executes a kernel is mainly organized as a grid of the thread blocks in the diagram of the GPU software architecture. Since limited element co-processors in a GPU, the number of the parallel computations may exceed the number of co-processors that is the same to the maximum number of the threads. So the threads must be synchronized and each batch of the computation is performed in turn. In other words, each step of the intensive calculation must be done in turn over the limited threads from the corresponding co-processors in the GPU. In the architecture of the NVIDIA's GPU, each thread is distinguished by its thread identity (ID) in each block which is also identified by its block ID in each grid. Then the thread ID of a thread with index (*td.x*, *td.y*, *td.z*) in a three dimension block (*Dx*, *Dy*, *Dz*) is indexed as (td.x+td.y$\times$*Dx*+td.z$\times$*Dx*$\times$*D*). Therefore, the multi-threads in the GPU can be executed for the parallel programs by the proper thread ID indexes.

B. Parallel QC LDPC decoding on GPU using CUDA

In the CUDA implementation of the log-BP algorithm, a thread is assigned to calculate the message of a variable node (VN) or a check node (CN) in the form of the logarithm likelihood ratios (LLRs). And a more simplified memory access version of the log-BP algorithm can be proposed due to the good feature of a regular QC LDPC matrix structure. So the workflow of the algorithm can be designed and shown in Fig. 1.

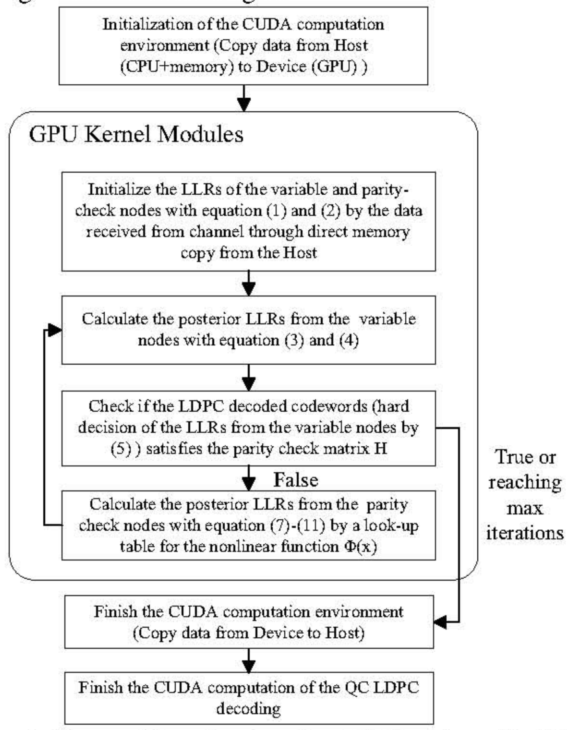
Initialization of the CUDA computation environment (Copy data from Host (CPU+memory) to Device (GPU))
GPU Kernel Modules
Initialize the LLRs of the variable and parity-check nodes with equation (1) and (2) by the data received from channel through direct memory copy from the Host
Calculate the posterior LLRs from the variable nodes with equation (3) and (4)
Check if the LDPC decoded codewords (hard decision of the LLRs from the variable nodes by (5)) satisfies the parity check matrix H
False
Calculate the posterior LLRs from the parity check nodes with equation (7)-(11) by a look-up table for the nonlinear function Φ(x)
True or reaching max iterations
Finish the CUDA computation environment (Copy data from Device to Host)
Finish the CUDA computation of the QC LDPC decoding

Fig. 1. The workflow for decoding LDPC codes with CUDA, where the kernel modules means the program executed on GPU in parallel with multi-threads.

From Fig. 1, the parallel CUDA implementation of the LDPC decoding is concluded as follows which is similar to that of [8]. However, the logarithm version of the LDPC decoding is adopted in our algorithm. And the efficient address of the messages by the properties of the QC LDPC matrix is proposed to further improve the parallel memory access capability to speed up the LDPC decoding.

Step 1. Copy data to the global memory in GPU.

Copy required prior *LLR*s of the input data for the LDPC decoding from host memory in PC to global memory in GPU. And the *LLR*s are executed by the CPU s as the basic calculation of the $L(c_i)$ in (1) where the $\{c_i\}$ sequences are the input raw data for decoding. So all threads can acquire the basic prior data in the global memory for computations. These *LLR* data include a structure array *S* containing $L(r_{ji})$ and $L(q_{ij})$ for calculating (3) and (11), two mapping arrays indicating the positions where VN *j* and CN *i* can access $L(r_{ji})$ and $L(q_{ij})$ in the array *S*. Another array of memory units is used to temporarily store the codewords.

Step 2. Initialize $L(r_{ji})$ and $L(q_{ij})$ in parallel by GPU.

Due to the lack of efficient calculation of division, (1) is calculated sequentially by the host CPU. Then the algorithm initializes the *LLR*s according to (2). Each thread is assigned to a variable node and computes all VN *LLR*s as $L(q_{ij})$ with the mapping array.

Step 3. Calculate and exchange messages (*LLR*s) from the check node to the variable node.

Compute $L(r_{ji})$ at the check nodes in parallel by (7)-(11). A thread is responsible for a particular check node *i*, and calculates all respective $\alpha_{i'j}$ in (8), $\beta_{i'j}$ in (9) and then $L(r_{ji})$ in (11) with a look-up-table like array to store the value of the complex function $\phi(x)$ which is calculated in advance. In addition, the shared memory on device is adopted to accelerate the massive number of looking up table for function value in the calculation of (11).

Step 4. Calculate current estimated codeword for judgment.

Following the steps of (4) and (5), *N* (code length) threads are assigned to calculate the codeword. If the estimated codewords satisfy (5) or the iterations reach the specified maximum iterations, finish the LDPC decoding for the codewords with success decoding or incomplete decoding and go to step 6 to output the codewords. Otherwise, carry out step 5 for further iterations.

Step 5. Calculate and exchange messages (i.e. *LLR*s) from the variable nodes to the check nodes.

Compute $L(q_{ji})$ at the variable nodes in parallel by (3) and the corresponding $L(Q_i)$ in (4) for further estimation of the input code words in (5). Each variable node is also assigned to a thread for parallel computation just as each check node does. So all threads can simultaneously calculate $L(q_{ji})$ and $L(Q_i)$ with high efficiency. After that, go to step 3 to continue the iterations.

Step 6. Output the codewords to the host.

The decoded codewords are copied from the GPU's global memory to the host memory for final output.

C. Architecture of the GPU CUDA platform for the QC LDPC decoding with efficient memory access technique

The architecture of the threads in the CUDA that executes a kernel is organized as a grid of thread blocks shown in Fig. 2. A batch of threads can cooperate by sharing data through the fast shared memory and synchronizing executions efficiently. So they have good features of high memory access bandwidth for huge parallel data throughput.

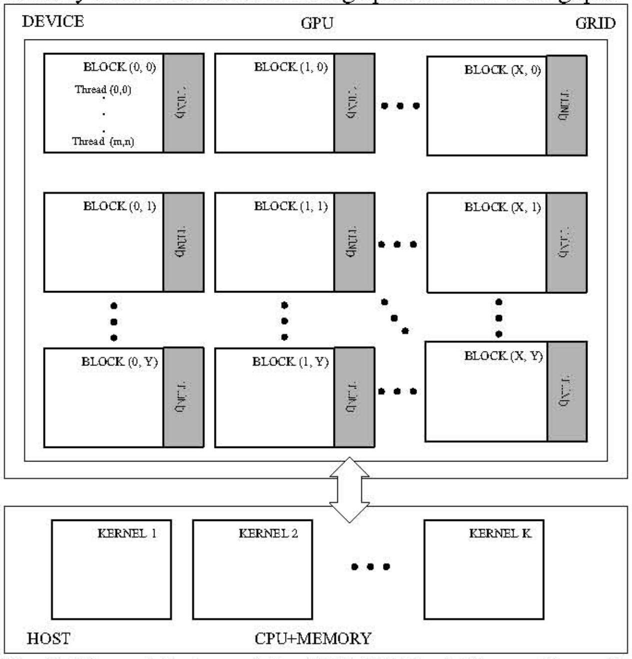

Fig. 2. The architecture of the GPU CUDA platform with multi-thread calculation [6].

In Fig. 2, the huge calculation power is mainly caused by the GPU with the multi-processor hardware and the corresponding multi-thread processing capability. The operation of an actual application must be modified as a parallel calculation structure to fit the GPU multi-threads calculation. Otherwise, the GPU power can be utilized for a program with serial structure and it even worse than the ordinary CPU-based scheme since the computation of a single processor in a GPU is much slower than that of an ordinary CPU and the access time of the global memory is also very longer than that between a CPU and a memory in a personal computer. From [6], if the calculations scale of an application is larger than all number of the threads in the GPU at a time, the parallel calculation must be arranged for several batches. Then the synchronization mechanism must be employed to guarantee the computation integrity. And it can be implemented with the subroutine "_syncthreads()" in CUDA programming. Since the calculation ability of each sub-processor in GPU is fixed, the computation efficiency of LDPC decoding mainly lies on the speed of the interface, i.e. the speed bottleneck between the co-processors and the fast share memory or the slow global memory. With the parallel nature of LDPC decoding mentioned in section II, the CUDA programming need special design of the data storage techniques to improve the memory access efficiency, especially the reduction of the access of the global memories in the GPU. And it can be implemented by using the quasi-cyclic structure of the QC LDPC parity-check matrix in the decoding iterations. The memory access scheme can be shown in Fig. 3.

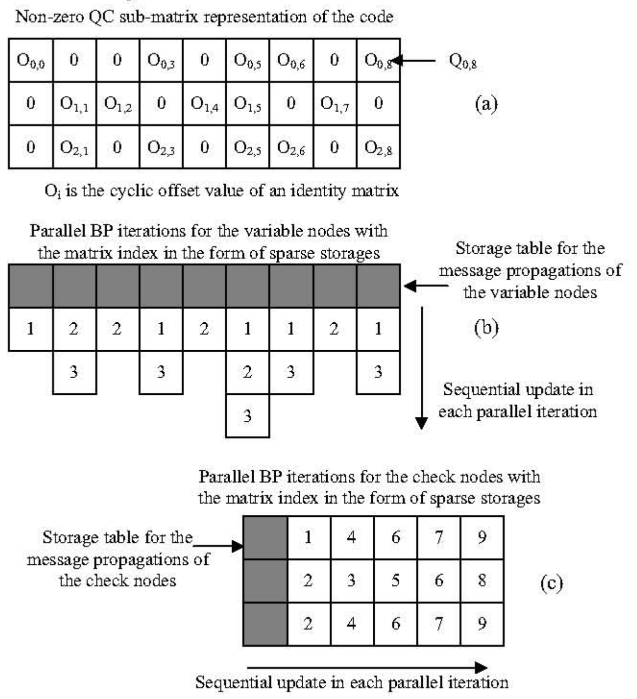
Non-zero QC sub-matrix representation of the code
O0,0 | 0 | 0 | O0,3 | 0 | O0,5 | O0,6 | 0 | O0,8
0 | O1,1 | O1,2 | 0 | O1,4 | O1,5 | 0 | O1,7 | 0
0 | O2,1 | 0 | O2,3 | 0 | O2,5 | O2,6 | 0 | O2,8
Q0,8
(a)
Oi is the cyclic offset value of an identity matrix
Parallel BP iterations for the variable nodes with the matrix index in the form of sparse storages
Storage table for the message propagations of the variable nodes
1 2 2 1 2 1 1 2 1
3 3 2 3 3
3
(b)
Sequential update in each parallel iteration
Parallel BP iterations for the check nodes with the matrix index in the form of sparse storages
Storage table for the message propagations of the check nodes
1 4 6 7 9
2 3 5 6 8
2 4 6 7 9
(c)
Sequential update in each parallel iteration

Fig. 3. The storage architecture of a QC LDPC code matrix in a GPU CUDA platform with multi-thread calculation [6].

The memory access approach can be described as follows. In Fig.3, The sparse parity-check matrix of a QC LDPC code is detonated as a basic matrix ($N\times N$) of the sub-matrix $Q_{i,j}$ with subscript row-i and column-j which is the circular permutation of an identity matrix. And the circular offset value of each sub-matrix $Q_{i,j}$ is $o_{i,j}$ shown in Fig.3 (a) respectively. Then the original complex QC LDPC matrix can be compactly represented with a small basic matrix shown in Fig.3 (a). So the updates of the messages in the variable and the check nodes can be described in Fig.3 (b) and Fig.3 (c) with the way of each sub-matrix node calculation and addressing in sequential. Suppose $o_{i,j}$ is the offset value of the sub-matrix, the message updates of the variable nodes can follow the equation (3) and the calculations are carried out according to each variable node in the basic matrix shown in Fig.3 (b). So the access to the variable and check nodes in the compactly stored memory can be illuminated as follows. Notice that both row and column sequence number, starting at 0, from the cyclic offset value $o_{i,j}$ with sub-matrix $Q_{i,j}$ ($N\times N$), the real row and column indexes of the k-th node in $Q_{i,j}$ are calculated as (12),

$$\begin{aligned} R &= (i\times N) + (N + o_{i,j} - k + 1) \bmod \mathrm{N} \\ C &= (j\times N) + (N - o_{i,j} + k - 1) \bmod \mathrm{N}, \end{aligned} \qquad (12)$$

where R and C is the actual row and column index of the k-th node in $Q_{i,j}$ in an LDPC code matrix. So the memory used to store the position of all “1” in the matrix is reduced to $1/N$ of that of randomly organized code matrix. Therefore, the memory access time can be reduced to $1/N$ too under the same memory access approach. Meanwhile, the messages from the variable and the check node can be stored in a shared memory of the GPU and the data can be exchanged with the original channel input information in batches to integrate the memory access processing.

Finally, the method of a CUDA based program code for QC LDPC decoding can be carried out as the way of the grid and the threads. Or it can be represented in the form of CUDA code such as “CUDALDPC_DEC <<< grid, threads >>> (message, matrix_N, matrix_M, matrix_data, channel_variance, …)”, where all parameters in the function CUDALDPC_DEC are input parameters of the algorithm in Section II. Therefore, the update of the messages from the variable nodes and the check nodes are executed alternatively with highly parallel iterations which can utilize the multi-threads processing of the GPU and greatly improve the calculation efficiency of LDPC decoding.

IV. NUNERICAL SIMULATIONS AND RESLUTS ANALYSES

By the efficient parallel simulation scheme proposed in section III, we can present the performance results of the parallel BP algorithm of LDPC decoding in a GPU based on the CUDA. The experiment is performed under a PC with Intel E5300 CPU of 2.6 GHz, 2GB 400MHz DDR memory, and a NVIDIA 9600GT GPU with 512MB graphic memory. The softwares are CUDA Toolkit 2.3, CUDA SDK 2.3 and CUDA driver 190.38. The low rate QC LDPC codes [10] are adopted for testing the performance of the code. The LDPC code length is 5120 (code rate 1/5), 6144 (code rate 1/6) and 10704 (code rate 1/6) respectively in order to investigate the influence of the computation scale on the performance of the same GPU platform. In the simulations, each E_b/N_0 will be run until a specified number (e.g. 100) of error frames occur or a total of 0.8 million trials have been run. The maximum iterations are 50 for each LDPC frame decoding.

With above parameters, the bit error rate (BER) results for all QC LDPC cases in ordinary CPU platform (serial C program codes) and the proposed GPU platform (parallel CUDA program codes) are almost similar. However, the speed of two scheme are different and the relative speed of each LDPC frame decoding measured in millisecond (ms) with three different case of code length is shown in Fig. 4. From Fig. 4, we can obtain the Speedup of GPU versus CPU for three different matrices by using 64 threads per block of NVIDIA 9600GT GPU card and it is shown in Fig. 5. From Fig. 4 and Fig. 5, the simulation of the case with LDPC code length 5120 and 6144 by the GPU is just a little faster than that by the GPU. However, the comparison is remarkable when the code length turns to 10704 where the former is about7 times of that of the CPU, which is much faster than the latter. So the speedup of the two simulation scheme with different LDPC code length shown in Fig. 5 indicate that much more efficiency can be improved with much longer code length. And the reasons are listed below. The memory access in the GPU, especially the global memory accessing, requires comparable time when compared to that of the parallel iteration of LDPC decoding. When the code length is short, the overhead of global memory access occupies quiet large portion of full execution time. However, when the code length grows large, the portion of memory access in all execution time decreases and the time consumption of LDPC decoding increases a lot. So we can get the conclusion as shown in Fig. 5 that more code length in LDPC decoding leads to more computation efficiency.

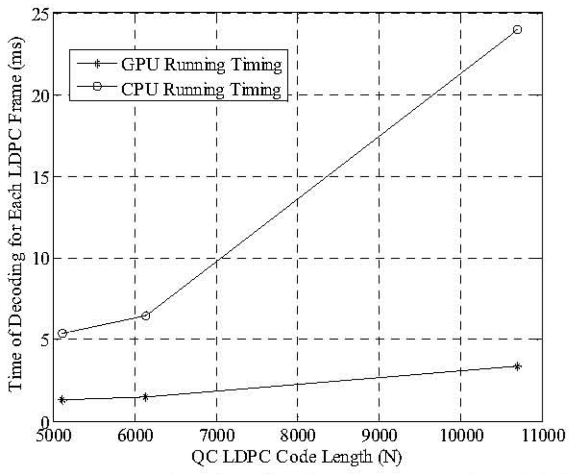

Fig. 4. Average running time for the QC LDPC codes with three different code length (5120. 6144 and 10704) by using 64 threads per block in the GPU.

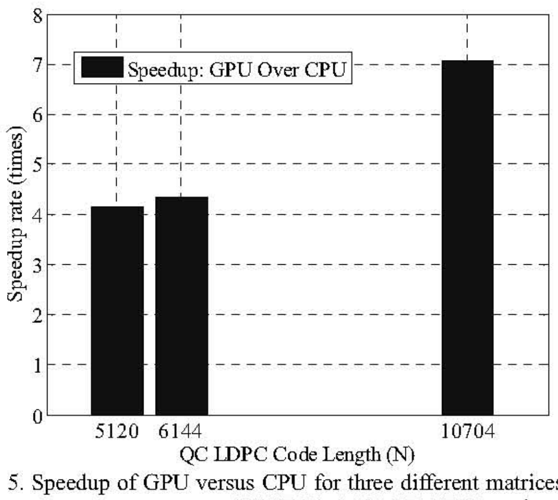

Fig. 5. Speedup of GPU versus CPU for three different matrices by using 64 threads per block of NVIDIA 9600GT GPU card.

Furthermore, if there are sufficient independent arithmetic instructions that are arranged for operation while the GPU is waiting for the global memory access, much of this global memory latency can be hidden by the thread scheduler just like an assembly line [7]. This result also verifies that CUDA is more suitable for compute-intensive, highly parallel computation where the time consumptions of the global memory access can be neglected in such computations.

V. CONCLUSIONS

An efficient parallel simulation scheme of QC LDPC decoding is proposed to accelerate simulation speed greatly. It employs a GPU to perform parallel simulation of LDPC decoding. Other than full hardware based schemes, it adopts lower costs and program complexity of CUDA technology. The GPU also provides parallel computation with highly multi-thread co-processors and very high memory bandwidth. Based on the sparse description and parallel processing of a QC LDPC decoding matrix, the proposed scheme updates all variable or check nodes in an LDPC decoding iteration simultaneously. In addition, the bottleneck of the simulation efficiency by the memory access is also discussed to explain the speedup among different LDPC code lengths. Therefore, the proposed method provides an efficient and fast approach of LDPC decoding and the error floors.

ACKNOWLEDGMENT

This work was supported by National Natural Science Foundation of China (No. 61001133, No. 61102066), China Postdoctoral Science Foundation (No. 2011M500999) and Open Project of Zhejiang Provincial Key Laboratory of Information Network Technology (No.201101).

REFERENCES

[1] R. G. Gallager, "Low-density parity-check codes," MIT Press, Cambridge, MA, 1963.

[2] D. J. MacKay. "Good error-correcting codes based on very sparse matrices". IEEE Transaction on Information Theory, 1999, 45(2), pp. 399-431.

[3] F. R. Kschischang, B. J. Frey and H. A. Loeliger, "Factor graphs and the sum-product algorithm", IEEE Trans. Inform. Theory, vol. 47, Feb 2001, pp. 498-519.

[4] C. Howland and A. Blanksby, "Parallel decoding architectures for low density parity check codes", The 2001 IEEE Int. Symp. on Circuits and Systems, 2001. ISCAS 2001. vol. 4, May 2001, pp. 742-745.

[5] Oh. Daesun, K.K Parhi, "Efficient highly-parallel decoder architecture for quasi-cyclic low-density parity-check codes", IEEE Int. Symp. on Circuits and Systems, 2007. ISCAS 2007. pp. 1855-1858, May 2007.

[6] NVIDIA Corporation. NVIDIA CUDA compute unified device architecture programming guide. V2.3, 2009.. http://www.nvidia.com/object/cuda_develop.html.[2009]

[7] G. Falcão, L. Sousa and V. Silva, "Massive parallel LDPC decoding on GPU", 13th ACM SIGPLAN Symp. On. Principles and Practice of Parallel Programming (PPoPP 2008), Feb. 20-23, 2008, pp. 83-90.

[8] S. Wang, S. Cheng and Q. Wu, "A parallel decoding algorithm of LDPC codes using CUDA", Proc. 42nd Asilomar Conf. on Signals, Systems, and Computers, Pacific Grove, CA, October 2008.

[9] J. Bao, Y. Zhan, J. Wu and J. Lu, "Design of efficient low rate QCARA GLDPC codes", Proc. 2009 IET Int. Commun. Conf. on Wireless Mobile & Computing (CCWMC 2009), Dec. 2009.