INTERNATIONAL RESEARCH JOURNAL IN ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET) E - ISSN: 2454-4752 P - ISSN : 2454-4744
VOL 2 ISSUE 3 (2016) PAGES 1041-1045
RECEIVED : 29/05/2016. PUBLISHED : 08/06/2016
June 8, 2016

REDUCING CODE LENGTH AND COMPUTATION TIME OF QC-LDPC DECODERS

[1]G.Vinothini, [2]K.Eswari,
[1]PG Scholar, Dept of VLSI Design, Roever Engineering College, Perambalur,
[2]Asst prof, Dept of ECE, Roever Engineering College, Perambalur.

Abstract:
Low density parity-check (LDPC) codes were invented by Robert Gallager but had been ignored for years until Mackay rediscovered them . They have attracted much attention recently because they can achieve excellent error correcting performance based on the belief propagation (BP)decoding algorithm. However, the BP decoding algorithm requires intensive computations. For applications like optical communication which requires BERs down to 10−15, using CPU-based programs to simulatethe LDPC decoder is impractical. Fortunately, the decoding algorithm possesses a high data-parallelismfeature, i.e., the data used in the decoding process are manipulated in a very similar manner and can be processed separately from one another.

Keywords – LDPC, BP, BER.

1. INTRODUCTION

A graphics processing unit (GPU) provides a parallel ar- chitecture which combines raw computation power with pro- grammability. GPU provides extremely high computational throughput by employing many cores working on a large set of data in parallel. In the field of wireless communi- cation, although power and strict latency requirements of real communication systems continue to be the main chal- lenges for a practical real-time GPU-based platform, GPU- based accelerators remain attractive due to their flexibility and scalability, especially in the realm of simulation acceleration and software-defined radio

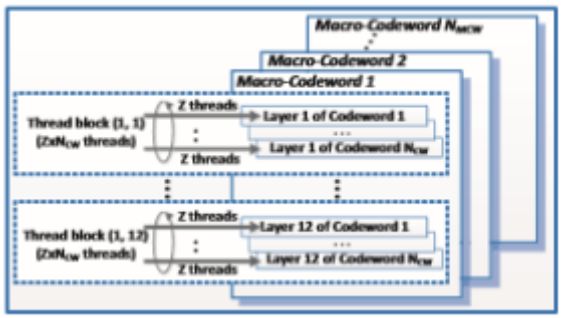


Fig.1. Multi-codeword parallel decoding algorithm.

(SDR) test-beds. Recently, GPU- based implementations of several key components of com- munication systems have been studied. For instance, a soft information multiple-input multiple-output (MIMO) detector is implemented on GPU and achieves very high throughput [1]. In [2], a parallel turbo decoding accelerator implemented on GPU is studied for wireless channels. Low-density parity-check (LDPC) decoder [3] is another key communication component and the GPU implementations of the LDPC decoder have drawn much attention recently, due to its high computational complexity. LDPC codes are a class of powerful error correcting codes that can achieve near- capacity error correcting performance. This class of codes are widely used in many wireless standards such as WiMax (IEEE 802.16e), WiFi (IEEE 802.11n) and high speed magnetic storage devices. The

flexibility and scalability make GPU a good simulation platform to study the characteristics of different LDPC codes or to develop new LDPC codes. Recently, parallel implementations of high throughput LDPC decoders are studied in [4]. In [5], the researchers optimize the memory access and develop parallel decoding software for cyclic and quasi-cyclic LDPC (QC-LDPC) codes. However, there is still great potential to achieve higher performance by developing better algorithm mapping according to the GPU's architecture.

2. LDPC DECODING ALGORITHM

The binary LDPC codes can be defined by the equation $H \cdot xT = 0$, in which x is a codeword and H is an $M \times N$ sparse parity check matrix. Quasi-Cyclic LDPC (QC-LDPC) codes are a special class of LDPC codes with a structured H matrix, which can be generated by the expansion of a $Z \times Z$ base matrix. As an example, Fig. 1 shows the parity check matrix for the (1944, 972) 802.11n LDPC code with sub- matrix size $Z = 81$. In this matrix representation, each square box with a label Ix represents an 81×81 circularly right- shifted identity matrix with a shifted value of x, and each empty box represents an 81×81 zero matrix. According to Equations, the decoding process can be split into two stages: the horizontal processing stage and the APP update stage. We can create one compu- tational kernel for each stage, which runs in the GPU. The host code running in the CPU takes charge of the CUDA initialization and memory copy between host and device. The early termination (ET) algorithm is used to avoid unnecessary computations when the decoder already converges to the correct codeword. For the LDPC codes, the parity check equations $H \cdot xT = 0$ can be used to verify the correctness of the decoded codeword. A new CUDA kernel with M threads is launched and each thread calculates one parity check equation independently. Since the decoded codeword x, compact H matrix and parity check results are used by all the threads, on-chip shared memory is used to speed up the memory access. After the concurrent threads finish computing the parity check equations, we reuse these threads to perform a reduction operation on all the parity check results to generate the final ET check result, which indicates the correctness of the codeword. For multi-codeword parallel decoding, we propose a tag-based ET algorithm. We assign one tag per codeword and mark the tag once the corresponding parity check equation is satisfied. Once the tags for all the codewords are marked, the iterative decoding process is terminated.

3. MAPPING LDPC

The decoding process can be split into two stages: the horizontal processing stage and the APP update stage. We can create one compu- tational kernel for each stage, which runs in the GPU. The host code running in the CPU takes charge of the CUDA initialization and memory copy between host and device. 1) CUDA Kernel 1: Horizontal Processing: During the horizontal processing stage, since all the CTV messages are calculated independently, we could use many parallel threads to process these CTV messages. For an $M \times N$ H matrix, M threads are spawned, and each thread processes a row. Since all non-zero entries in a sub-matrix of H have the same shift value (one square box in Fig. 1), threads processing the same layer (a row of square boxes in Fig. 1) have almost exactly the same operations when calculating the CTV messages. Msub thread blocks are used and each consists of Z threads. Taking the 802.11n (1944, 972) LDPC code as an example, 12 thread blocks are generated, and each contains 81 threads, so there are a total of 972 threads used to calculate the CTV messages. 2) CUDA Kernel 2: APP value update: During the APP update stage, there are N APP values to be updated. Similarly, the APP value update is independent among variable nodes. Thus, Nsub thread blocks are used, with Z threads in each thread block. In the APP update stage, there are 1944 threads which are grouped into 24 thread blocks working concurrently for the 802.11n

INTERNATIONAL RESEARCH JOURNAL IN ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET) E - ISSN: 2454-4752 P - ISSN : 2454-4744
VOL 2 ISSUE 3 (2016) PAGES 1041-1045
RECEIVED : 29/05/2016. PUBLISHED : 08/06/2016
June 8, 2016

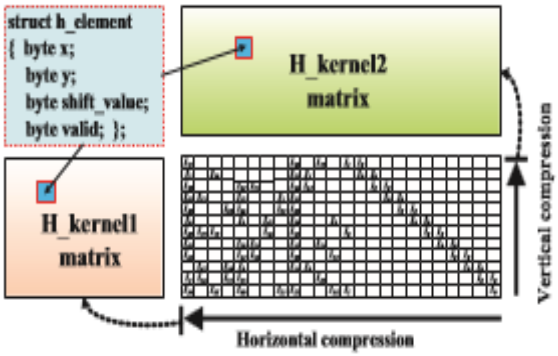
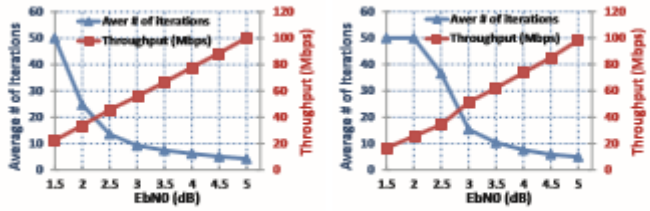


Fig.2. The compact representation for H matrix

(1944, 972) LDPC code. Kernel 2 finally makes a hard decision for each bit. Since the number of threads and thread blocks are lim- ited by the dimensions of the H matrix, multi-codeword decoding is needed to further increase the parallelism of the workload. A two-level multi-codeword scheme is designed. NCW codewords are first packed into one macro-codeword (MCW). Each MCW is decoded by a thread block and NMCW MCWs are decoded by a group of thread blocks. The multi- codeword parallel decoding algorithm is described in Fig. 2. Since multiple codewords in one MCW are decoded by the threads within the same thread block, all the threads follow the same execution path. Moreover, the latency of read-after-write dependencies and memory bank conflicts can be completely hidden by a sufficient number of active threads.

4. EXPRIMENTAL RESULTS

The experimental setup to evaluate the performance of the proposed architecture on the GPU consists of an NVIDIA GTX470 GPU with 448 stream processors, running at 1.215GHz and with 1280MB of GDDR5 device memory. We implement both the log-SPA and the min-sum algorithm. Assume the codeword length is Nbits , the total number of codewords is Ncodeword , the simulation number is NSim , and the running time is Ttotal , which contains both the decoding time and the time for memory copy between host and device. The throughput can be calculated by: Throughput = (Nbits × NSim × Ncodeword)/T total . According to the capacity of GTX470 GPU, around 300 codewords are processed in parallel in the multi-codeword decoding scheme (Ncodeword = 300). Table I shows the throughput of our implementation for both the 802.11n code and WiMAX code with different number of iterations (Niter). The throughput for the log-SPA algorithm is comparable to the min-sum algorithm. The reason is that GPU implementation employs very efficient intrinsic functions logf() and expf() . And the bottleneck for GPU implementation is in the long latency of the device memory access, therefore, the run time for the extra instructions in the log-SPA is hidden behind the memory access latency. these two codes are similar, which means that the computa- tional workload is comparable. Therefore, the WiMAX code which has longer codewords tends to have higher throughput according to the throughput equation. Furthermore, there are
more arithmetic instructions per memory access for a longer codeword, which can hide the memory access overhead. the throughput results and the average number of iterations with the parallel early termination (ET) scheme. As the SNR (represented by EbN0) increases, the average number of iterations decreases and the decoding throughput increases. that the parallel early termination scheme significantly speeds up the simulation for the high SNR



(a) Log-SPA algorithm (b) Scaled min-sum algorithm

. For low SNR, the ET version may be slower than the non-ET version due to overhead of the ET kernel. Therefore, an adaptive scheme can be used to speed up the simulation for the whole SNR range – the ET kernel launches only when the simulation SNR is higher than a specific threshold. It is difficult to use massive threads to fully occupy the computation resources of the GPU when decoding the irregular LDPC codes. When processing an irregular LDPC code, im- balanced workloads cause the threads on GPU to complete the computations at different times and runtime is bounded by the threads with the most amount of work. Table II compares our work with the related work. Table II shows that although the irregular codes we used are theoretically harder to get higher throughput than the ones in the related work, our decoder still outperforms the related work with significant improvement, especially when the parallel ET scheme is used. Our work is directly comparable to [5] since they also implemented a decoder for 802.11n (1944, 972) QC-LDPC code. Although the GPU used in this work has approximately twice the amount of computation resource as in [5], our decoder achieves more than 50 times throughput compared to their work. This huge improvement can be attributed to our highly optimized algorithm mappings, efficient data structures and the memory access optimizations.

CONCLUSION

This paper presents the techniques and design methodology to fully utilize a GPU's computational resources to accelerate a computation-intensive DSP algorithm. As a case study, a massively parallel implementation of LDPC decoder on GPU is presented. To achieve high decoding throughput, several techniques including efficient algorithm mapping, compact data structures and memory access optimizations are em- ployed. We take the LDPC decoder for the IEEE 802.11n WiFi LDPC code and 802.16e WiMAX LDPC code as examples to demonstrate the performance of our GPU-based implementa- tion. The simulation results exhibit that our LDPC decoder can achieve high throughput around up to 100.3Mbps.

REFERENCES

[1] M. Wu, Y. Sun, S. Gupta, and J. Cavallaro, "Implementation of a high throughput soft MIMO detector on GPU," Journal of Signal Processing Systems , pp. 1–14, 2010, 10.1007/s11265-010-0523-4. [Online]. Available: http://dx.doi.org/10.1007/s11265-010-0523-4

[2] M. Wu, Y. Sun, and J. Cavallaro, "Implementation of a 3GPP LTE turbo decoder accelerator on GPU," in IEEE Workshop on Signal Processing Systems (SIPS) , 2010, pp. 192 –197.

[3] R. Gallager, "Low-density parity-check codes," IRE Transactions on Information Theory , vol. 8, no. 1, pp. 21 –28, 1962.

[4] G. Falcao, L. Sousa, and V. Silva, "Massively LDPC decoding on multicore architectures," IEEE Transactions on Parallel and Distributed Systems , vol. 22, no. 2, pp. 309 –322, 2011. [5] H. Ji, J. Cho, and W. Sung, "Memory access optimized implementation of cyclic and quasi-cyclic LDPC

codes on a GPGPU," Journal of Signal Processing Systems , pp. 1–11, 2010, 10.1007/s11265-010-0547- 9. [Online]. Available: http://dx.doi.org/10.1007/s11265-010-0547-9

[6] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," IEEE Transactions on Communications , vol. 47, no. 5, pp. 673 –680, May 1999.

[7] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," IEEE Transactions on Communications , vol. 53, no. 8, pp. 1288 – 1299, 2005.

[8] S.-H. Kang and I.-C. Park, "Loosely coupled memory-based decoding architecture for low density parity check codes," in IEEE Custom Integrated Circuits Conference (CICC) , 2005, pp. 703 – 706.

`