FFT-SPA NON-BINARY LDPC DECODING ON GPU

J. Andrade, G. Falcao, V. Silva*

Instituto de Telecomunicações
Dept. of Electrical and Computer Eng.
University of Coimbra, Portugal

Kenta Kasai

Dept. of Comm. and Integrated Systems
Graduate School of Science and Eng.
Tokyo Institute of Technology, Japan

ABSTRACT

It is well known that non-binary LDPC codes outperform the BER performance of binary LDPC codes for the same code length. The superior BER performance of non-binary codes comes at the expense of more complex decoding algorithms that demand higher computational power. In this paper, we propose parallel signal processing algorithms for performing the FFT-SPA and the corresponding decoding of non-binary LDPC codes over GF(q). The constraints imposed by the complex nature of associated subsystems and kernels, in particular the Check Nodes, present computational challenges regarding multicore systems. Experimental results obtained on GPU for a variety of GF(q) show throughputs in the order of 2 Mbps, which is far above from the minimum throughput required, for example, for real-time video applications that can benefit from such error correcting capabilities.

***Index Terms*—** Non-binary LDPC codes, GF(q), Communications, Error correcting codes, GPU

1. INTRODUCTION

Robert Gallager proposed Low-Density Parity-Check (LDPC) codes in 1962 [1], which allow capacity-approaching Bit Error Rate (BER) performances. While binary LDPC codes are defined over a GF(2), non-binary LDPC codes, defined over GF(q), in particular the binary extension field, i.e. $q = 2^m$ and $m > 1$, are known for having superior BER performance for the same code parity-check matrix binary image [2].

Mainly due to high computational power requirements on the decoder and to large memory demands imposed by the algorithm, few applications already incorporate the use of non-binary LDPC codes, in spite of their high potential. Compared to its binary counterpart, the conventional algorithms for decoding non-binary LDPC codes are more complex and require a substantial higher amount of memory for storing messages. Two application examples are Quantum Key Distribution (QKD), a cryptographic primitive that applies quantum mechanics for establishing secure communications, and video transmission systems that are capable of working over the erasure channel under severe Signal-to-Noise Ratio (SNR) conditions.

Although we have assisted to the recent development of binary LDPC decoders on Graphics Processing Units (GPUs) [3, 4, 5, 6, 7, 8], the importance of the non-binary case seems to be underestimated yet. In this paper, we propose a parallel decoder based on the Fast-Fourier Transform Sum-Product Algorithm (FFT-SPA) that exploits the multithread capabilities of the GPU to parallelize the intensive computation of Variable Nodes (VNs) and, more importantly, of Check Nodes (CNs). We propose four major approaches to balance processing and accelerate their decoding performance on GPUs: i) we show that the size of data structures involved allows optimizing the Tanner graph representations and Fast-Fourier Transforms (FFTs) indexing accesses by exploiting the shared and constant memories of the GPU system, which are fast and tightly coupled to the processing cores, instead of using the slow global memory; ii) we perform a GPU-aware optimization of the more intensive kernels, namely the FFT, mainly based on the intensive use of register and shared memory; iii) in order to minimize the number of FFT calls, which are computationally intensive, we compute three sub-kernels (normalization of probability distribution, syndrome calculation and decoded bits decision) in the Fourier domain; and iv) we use multicodeword data-parallelism to increase the level of workload distribution of the kernels and, therefore, allow the hardware scheduler to have more opportunities to mask memory accesses with computation [8].

2. NON-BINARY LDPC CODES AND DECODING COMPUTATIONAL COMPLEXITY

Non-binary LDPC codes are defined by a parity-check matrix $\mathbf{H}$ in GF(q), composed by M rows and N columns. The Tanner graph [9] defines each row in $\mathbf{H}$ as a CN and each column as a VN. CNs and VNs are connected when a non-null element exists in $\mathbf{H}$. The edges sets $C(v)$ and $V(c)$ define the edges connected to CN_v and to VN_c, respectively, and their cardinality the CN and VN weight, denoted by d_c and d_v.

*This work is supported by Instituto de Telecomunicações and Fundação para a Ciência e Tecnologia under grants PEst-OE/EEI/LA0008/2011 and SFRH/BD/78238/2011, through the European programs POPH-QREN and FSE. Kenta Kasai's work is supported by the Storage Research Consortium.

2.1. Decoding Algorithms Overview

The Sum-Product Algorithm (SPA) can be generalized to GF(q), leading to a higher complexity for the VN processing and in particular for the CN processing that follows a $O(M \times d_c \times 2^{q \times d_c})$ numerical complexity, due to the convolution term [10]. Fortunately, through the use of the FFT transform, convolutions can be expressed in terms of products, and the overall numerical complexity of the SPA can be lowered to $O(M \times d_c \times q \times 2^q)$ [10]. Other decoding algorithms address the CN processing computational complexity and try to ameliorate it, such as the Log-Fourier SPA (LFSPA), Extended Min-Sum (EMS) algorithm and Min-Max algorithm [10, 11, 12].

2.2. FFT-Sum Product Algorithm

The SPA can be resumed in four steps: *i)* the *probability mass function* (*pmf*) $p_v(x)$ is computed by the demodulator for all the received symbols, and all m_{vc} messages are initialized as in (1); *ii)* the CN processing reads the $m_{vc}^{(i)}$ messages from their adjacent VNs and computes $m_{cv}^{(i+1)}$ messages according to (2); *iii)* the VN processing reads the $m_{cv}^{(i+1)}$ messages from their adjacent CNs and computes $m_{vc}^{(i+1)}$ messages according to (3); *iv)* an *a-posteriori* likelihood $m^{*(i+1)}_v$ can be inferred from the $m_{vc}^{(i+1)}$ messages (4) and hard-decoded to the symbol with highest likelihood (5). The algorithm iteratively executes steps *ii)* to *iv)* until a maximum number of iterations is reached or a valid codeword is decoded:

$$p_v(x) = m_{vc}^{(0)}(x) = p(v_v = x|y_v) \tag{1}$$

$$m_{cv}^{(i)}(x) = \sum_{\mathbf{v}:v_v=x} p(z_c = 0|\mathbf{v}) \prod_{v' \in V(c)\setminus c} m_{v'c}(x) \tag{2}$$

$$m_{vc}^{(i)}(x) = k_{vc} p_v(x) \prod_{c \in C(v)\setminus c'} m_{c'v}(x) \tag{3}$$

$$k_{vc} \Leftarrow \sum m_{vc}^{(i)}(x) = 1$$

$$m^{*(i)}_v(x) = K_v p_v(x) \prod_{c \in C(v)x} m_{vc}(x) \tag{4}$$

$$K_v \Leftarrow \sum m^{*(i)}_v(x) = 1$$

$$\hat{v}_n = \arg\max_x \{m^*{}_v(x)\}, \tag{5}$$

where (i) represents the i-th iteration. The convolution term in (2) can be simplified by moving to the Fourier domain, replacing (2) with (6). This involves computing the FFT of the m_{vc} messages prior to the CN processing and the FFT^{-1} of the m_{cv} messages afterwards.

$$m_{cv}^{(i)}(x) = \text{FFT}^{-1}\left\{ \prod_{v' \in V(c)\setminus c} \text{FFT}\left\{m_{v'c}(x)\right\} \right\} \tag{6}$$

Another useful property of the FFT usage is the ability to normalize the m_{vc} messages *a-posteriori*, namely in the CN processing at iteration $(i+1)$, instead of doing it in the VN processing at iteration (i). Since the first term of the FFT is the sum of all q likelihoods, i.e. k_{vc}, each output element of the FFT is divided by the first element.

Before and after the main CN processing (2) and (6), a *pmf* permutation and de-permutation is required, since VNs participating in a parity-check equation are multiplied by elements in GF(q) [13]. One of the benefits associated with the FFT-SPA decoding is that all arithmetic operations are defined in $\mathbb{R}$ [13]. The only dependency on GF(q) arithmetic is in the syndrome computation, which is independent of the decoding algorithm.

3. PROPOSED PARALLEL NON-BINARY LDPC DECODER

In this section the proposed FFT-SPA implementation and optimizations on the GPU are detailed.

Efficient Fast Hadamard Transform: The domain change operated in the CN processing is possible by using the Fast Hadamard Transform (FHT). The paramount importance of the use of the FHT to reduce the numerical complexity of CN updating is well illustrated in [10, 14]. However, the implementation details of the FHT present challenges in the context of the non-binary LDPC decoding problem, for the GPU architecture, especially for higher order GF(q), such as GF(256) and GF(128). We are thus motivated into developing a FHT for the non-binary LDPC decoding problem, efficiently mapping the algorithm onto the GPU shared memory architecture and suiting the LDPC decoder data structures. In [15], the authors have defined a GF(q) radix-2 FHT, which does not exploit the shared memory capabilities of the GPU engine, henceforth designated as *FHTq*. For the latter purpose, a GF(256) radix-2 FHT was devised, which exploits the shared memory architecture and minimizes the number of shared memory bank conflicts [16], designated *FHT256*. However, shared memory bank conflicts can be completely eliminated, by employing radix-n FHTs [17] tuned to the underlying GPU memory engine, designated as *FHTo*. Due to the finite field representation, input elements must be reordered. For instance, for GF(8) generated by the primitive polynomial $f(x) = x^3 + x + 1$, elements would be reorganized as follows: $(1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6) \rightarrow (0, 1, \alpha, \alpha^2, \alpha + 1, \alpha^2 + \alpha, \alpha^2 + \alpha + 1, \alpha^2 + 1)$, which in binomial form can be written as $(000, 001, 010, 100, 011, 110, 111, 101)_2 = (0, 1, 2, 4, 3, 6, 7, 5)_{10}$, where α is the primitive element, i.e. $f(\alpha) = 0$. Naturally, the output elements are organized to revert the former order.

We have integrated the efficient *FHTo* for GF(128) and GF(256) on the proposed decoder, since LDPC decoding over such high order Galois fields is much more complex than over lower orders. The former uses a mixed radix FHT, deploying a stage in radix-8 followed by two stages in radix-4, while

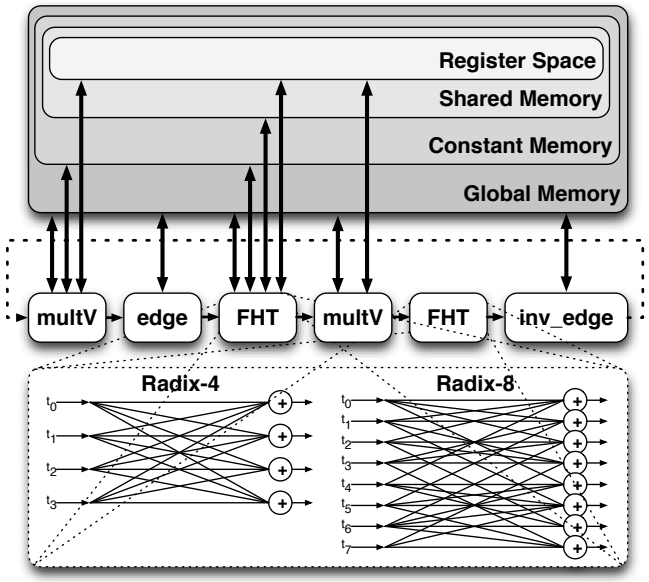

Fig. 1. FFT-SPA employment of the GPU memory hierarchy, also showing the radix-4 and radix-8 FHT. The most intensive FHT kernel takes full advantage of the memory hierarchy of the GPU, by exploiting the fast register space and shared memory for computation and data buffering.

the latter deploys a radix-4 implementation. Both kernels rely on 64 thread blocks not only to keep a reasonable number of threads active in the GPU engine, but also to limit the shared memory usage, a constraint to the number of active threads. Regardless of the design dimension, the developed *FHTo* reorganizes data according to the polynomial indexing at the input and at the output, given by the primitive polynomial, and also requires a dyadic to sequential reorganization, since its outputs come in dyadic ordering. The *FHTo* for GF(128) and GF(256) are depicted in Figure 1. A main design drive of the *FHTo* is to limit the usage of global memory to a minimum, thus using the shared memory to cache any intermediate values and the fast register space for computation.

Data Layout: One of the key challenges in LDPC decoding on the GPU architecture is how to devise efficient memory layouts and data structures which maximize the GPU memory engine bandwidth. The Tanner graph imposes non-sequential memory transactions between the messages exchanged between VNs and CNs [6]. However, if the binary image of the LDPC code is of relatively small dimensions, we are able to index the message-passing between nodes through Lookup Tables (LUTs) fitting in the 64KB constant memory. In the proposed decoder, a row-major transformation of the non-null elements in $\mathbf{H}$ suffices to index the loading and storing of m_{cv} and m_{vc} messages, respectively. This leaves the more complex CN processing without requiring LUT-reading of memory locations. Moreover, for GF(q) *pmf*, the requirement to store q likelihood values does not add any relevant complexity to the data layout. Elements are stored contiguously, with each consecutive q elements organized in ascending primitive element order , i.e. $m_{cv}(x) = \{m_{cv}(0), m_{cv}(1), \cdots, m_{cv}(\alpha^{q-2})\}$.

Coalesced Data Representation and Parallelism: Data representation is critical for LDPC decoding. Using GPU processors, quantization effects can be avoided, since they are optimized for IEEE-compliant floating-point arithmetic, that allows achieving peak bandwidths in the TFLOPs range. Thus, likelihoods are represented in single precision floating-point, and in order to fully exploit the bandwidth provided by the GPU memory engine we have defined an intra-thread data-parallelism level of 4, i.e. 4 codewords are loaded into the GPU at once. This way, we are able to achieve fully coalesced 128-bit aligned memory accesses, which help to overlap memory latency with computation, a key factor for extracting the best performance.

Algorithm 1 GF(q) FFT-SPA decoder on GPU

1: (**construct_from_file**) Load LDPC code and interleaver
2: (**InitEncoding**) Load arithmetic LUTS for GF(2^m) and FHT LUTs
3: (CPU to GPU memory transfer) Copy LUTs and interleaver
4: (**Encoding**) Generate a new codeword and random noise
5: (CPU→GPU) Copy codeword and random noise from the CPU to the GPU
6: (***gpu_initialize_message***) Transmit codeword through AWGN channel
7: (***init_mcv***) Initialize m_{cv} messages
8: **while** $\mathbf{c} \cdot \mathbf{H} \neq \mathbf{0}$ or maximum iterations reached **do**
9: - - - - - - - - - - (Start soft-decoding)
10: (***multV***) Execute the VN kernel
11: (***edge***) Permute m_{vc} messages
12: (***FHT***) Execute the FFT for m_{vc} messages
13: (***multC***) Execute CN kernel on the Fourier-domain
14: (***FHT***) Execute the FFT for m_{cv} messages
15: (***inv_edges***) De-permute the m_{cv} messages
16: - - - - - - - - - - (End soft-decoding, start hard-decoding)
17: (***deviceP_VN***) Compute the *a-posteriori* m^*_{vc} messages
18: (***FHT***) Execute the FFT for m^*_{vc} messages
19: (***decideB***) Hard-decode the m^*_{vc} messages into binary q-tuples
20: (***decideS***) Convert the binary q-tuples into GF(q) symbols
21: - - - - - - - - - - (End hard-decoding, start parity-check)
22: (***decideSynd***) Compute the syndrome vector
23: (GPU→CPU) Copy syndrome from the GPU to the CPU
24: **EvaluateSyndrome** Test if syndrome is the null vector.
25: - - - - - - - - - - (End parity-check)
26: **end while**

4. EXPERIMENTAL RESULTS

The proposed GPU FFT-SPA decoder, whose pseudo-code implementation is defined in Algorithm 1, was tested for a numerous set of parameters, which enables us to compare the different optimization steps carried out, and their impact on the overall performance of the decoder.

Table 1. Execution time and throughput of the FFT-SPA LDPC decoder for variable GF(q) and different FHT implementations. Numerals marked with * represent values obtained for the *FHT256*, † are obtained for the *FHTo* and all others for the *FHTq*.

Galois Field	GF(32)			GF(64)			GF(128)			GF(256)		
Iterations	5	10	15	5	10	15	5	10	15	5	10	15
Throughput [Mbit/s]	1.63	0.82	0.55	0.78	0.39	0.26	0.42	0.21	0.14	0.26	0.13	0.08
										1.52*	0.77*	0.52*
							3.34†	1.67†	1.11†	2.37†	1.20†	0.79†
Execution Time [ms]	7.51	14.93	22.38	15.80	31.55	47.41	29.01	58.02	86.88	48.27	96.43	145.16
										8.07*	15.90*	23.81*
							3.67†	7.36†	11.01†	5.20†	10.28†	15.51†

4.1. Apparatus

The decoder was profiled on an Asus P6T7 WS with an Intel i7 950 Central Processing Unit (CPU), a Tesla C1060 Nvidia GPU and 12GB of RAM, running GNU/Linux 3.4.4 and CUDA 5.0. The decoder was tested for $q \in \{32, 64, 128, 256\}$ for a regular $(2, 3)$-LDPC code with a binary image of $N = 384$ VNs and $M = 256$ CNs [2]. The execution times profiled measure only the soft-decoding kernels highlighted in Algorithm 1, and were obtained with CPU timers.

4.2. Throughput performance of non-binary LDPC decoders on the GPU

Experimental results for the relative weight of the developed GPU kernels are shown in Table 2. It can be seen that the *FHTq* implementation consumed 94.3% of the kernels execution time for GF(256). A significant improvement was achieved by using the *FHT256*, which lowered the FHT weight to 65.1%. The optimized *FHTo* further lowered it to 45.4% of the total execution time.

By using the *FHT256* and the *FHTo*, throughput is raised, at 5 iterations, from 0.26 Mbit/s to 1.52 and 2.37 Mbit/s, respectively for GF(256). For the GF(128) case, the throughput at 5 iterations was elevated from 0.42 Mbit/s to 3.34 Mbit/s. This is equivalent to speedups of 9.11x and 7.95x when compared to the *FHTq* and the *FHTo*.

Table 2. Relative weight (in percentage) of the FHT implementation in the execution time for GF(256).

Kernels	*FHTq*	*FHT256*	*FHTo*
FHT	94.3	65.1	45.4
multC	1.7	10.6	16.3
multV	1.6	10.1	15.8
edge	1.2	7.2	11.4
inv_edges	1.2	7.0	11.1

5. RELATED WORK

To the best of our knowledge, this is the first FFT-SPA implementation for GF(q), and has been preceded by two works on non-binary LDPC decoding on GPUs. Very recently, Wang *et al.* [14] have developed a GPU-based Min-Max LDPC decoder for GF(q). Their OpenCL implementation explores similar design space guides. However, given the specifications of the author's selected algorithm, some optimizations carried out in [14] are orthogonal to those proposed in this paper. Namely, the Forward and Backward optimization is applicable only to the Min-Max algorithm, as the FHT is to the FFT-SPA. Romero and Chang [18], present a non-binary LDPC decoder on GPU, where a sequential decoder exploits the parallelism inherent in the GF(q) representation. This work is tightly attached to the efficient development and mapping of a decoding schedule, and finding a trade-off between the lower throughputs per iteration inherent to sequential decoders and the lower number of iterations required to successfully decode a sequence. Hence, we find that the related works [14, 18] and the work proposed in this paper contribute to the non-binary LDPC decoding field on parallel multicore architectures in complementary ways.

6. CONCLUSIONS

In this paper we propose an efficient GPU FFT-SPA decoder for LDPC codes defined over GF(q), which exploits several aspects of the GPU architecture. Namely, efficient coalesced data structures have been devised which maximize the memory engine delivered bandwidth through proper alignment and higher levels of data-parallelism; a non-binary LDPC-oriented FHT has been developed which increased the decoding throughput to the Mbit/s range; efficient Fourier domain simplifications have been carried out in the FFT-SPA which reduce the complexity of the CN processing. The achieved decoder throughput and execution time prove the suitability of the GPU engine to the non-binary LDPC decoding problem. Moreover, the FFT-SPA achieves decoding throughputs which are competitive with results found in the literature, even considering that in our case the decoding complexity is superior and that higher orders of GF(q) have been tested.

7. REFERENCES

[1] R. G. Gallager, "Low-Density Parity-Check Codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[2] Charly Poulliat, Marc Fossorier, and David Declercq, "Design of Regular (2,dc) -LDPC Codes over GF (q) Using Their Binary Images," *IEEE Transactions on Communications*, vol. 56, no. 10, pp. 1626–1635, 2008.

[3] K.K. Abburi, "A Scalable LDPC Decoder on GPU," in *Proceedings of the 24th International Conference on VLSI Design*, Jan 2011, pp. 183–188.

[4] 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*, vol. 64, no. 1, pp. 149–159, 2011.

[5] M. Wu, S. Gupta, Yang Sun, and J. R. Cavallaro, "A GPU implementation of a real-time MIMO detector," in *Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS'09)*, October 2009, pp. 303–308.

[6] 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.

[7] G. Falcao, J. Andrade, V. Silva, and L. Sousa, "Real-time DVB-S2 LDPC decoding on many-core GPU accelerators," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2011, pp. 1685–1688.

[8] G. Falcao, V. Silva, L. Sousa, and J. Andrade, "Portable LDPC Decoding on Multicores Using OpenCL," *IEEE Signal Processing Magazine*, vol. 29, no. 4, pp. 81–109, July 2012.

[9] R. Tanner, "A Recursive Approach to Low Complexity Codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[10] D. Declercq and M. Fossorier, "Decoding algorithms for nonbinary ldpc codes over gf," *Communications, IEEE Transactions on*, vol. 55, no. 4, pp. 633 –643, april 2007.

[11] Kenta Kasai and Koichi Sakaniwa, "Fourier Domain Decoding Algorithm of Non-Binary LDPC Codes for Parallel Implementation," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2011*, Prague, Czech Republic, 2011, pp. 3128–3131.

[12] Valentin Savin, "Min-Max decoding for non binary LDPC Codes," in *ISIT*, Toronto, Canada, 2008, pp. 960–964.

[13] Rolando Antonio Carrasco and Martin Johnston, *Non-Binary Error Control Coding for Wireless Communication and Data Storage*, Wiley, Chichester, 2008.

[14] Guohui Wang, Hao Shen, Bei Yin, Michael Wu, Yang Sun, and Joseph R. Cavallaro, "Parallel nonbinary LDPC decoding on GPU," in *the 46th Asilomar Conference on Signals, Systems and Computers*, Nov. 2012.

[15] Y. Fujisaka K. Kasai and M. Onsjo, "Fft-based parallel decoder of non-binary ldpc codes on gpu: Kfo_nbldpc_gpu," [Online; accessed November/2012].

[16] Kenta Kasai Mikael Onsjo and Osamu Watanabe, "CUDA Implementation of Iterative Updating: the Radix-2 Algorithm and Discrete Fourier Transforms," 2010.

[17] Apple, "OpenCL_FFT," in *https://developer.apple .com/library/mac/#samplecode/OpenCL_FFT/ Introduction/Intro.html*, [Online; accessed Oct. 2012].

[18] David Romero and Nicholas Chang, "Sequential decoding of non-binary ldpc codes on graphics processing units," in *Signals, Systems and Computers (ASILOMAR), 2012 Conference Record of the Forty Sixth Asilomar Conference on*, Nov. 2012.