MULTICORE IMPLEMENTATION OF LDPC DECODERS BASED ON ADMM ALGORITHM

Imen DEBBABI, Nadia KHOUJA, Fethi TLILI

SUP'COM, GRESCOM Lab,
University of Carthage, Tunisia

Bertrand LE GAL, Christophe JEGO

Bordeaux INP, IMS Lab,
University of Bordeaux, France

ABSTRACT

Alternate direction method of multipliers (ADMM) technique has recently been proposed for LDPC decoding. Even though it improves the error rate performance compared with traditional message passing (MP) techniques, it shows a higher computation complexity. In this article, the ADMM decoding algorithm is first described. Then, its computation complexity is analyzed. Finally, an optimized version which benefits from the multi-core processors architecture as well as the ADMM algorithm's parallelism is presented. The optimized version of the ADMM decoder can achieve up to 30 Mbps for standardized LDPC codes on a laptop x86 processor. Therefore, it could guide an efficient GPU implementation for real-time and high-throughput decoding systems requiring correction performances beyond MP-Sum Product Algorithm (SPA) capabilities.

Index Terms— LDPC decoding, ADMM algorithm, muti-core, software optimization, Forward Error Correction codes.

1. INTRODUCTION

Low Density Parity Check (LDPC) codes are a class of rediscovered valuable linear block codes [1] [2]. They have gained widespread attention in various fields of wireless communications and are adopted as error-correcting codes by many digital communication standards such as DVB-S2 [3], DVB-S2X [4], WiMAX [5], WiFi [6] and WRAN [7]. Using linear programming (LP) optimizations, Feldman *et al.* in [8] proposed a novel algorithm for LDPC decoding. They showed that the error correction performance of this LP based decoder is comparable to and often better than the iterative MP decoding [9]. However, LP techniques are too complex in terms of computations and memory requirements [9–12] to be applicable to medium and long LDPC codes even with low-throughput requirement.

With the advance of the ADMM technique [13], a significant improvement towards LP LDPC decoding scalability and optimization is possible. Barman *et al.* in [14] presented a *two-slice* characterization for the parity polytope and developed an efficient Euclidean projection algorithm which is required for ADMM-based LP decoding. Despite the fact that ADMM is distributed and has strong convergence guarantees, current implementations of the ADMM-based LP decoders do not meet throughput standard requirements [15] [16]. Actually, even though the algorithm's complexity seems comparable to the MP decoding approach, the projection on the parity polytope, which is the convex hull of all the binary vectors with an even number of ones, still remains a huge time-consuming task even when compared with $atan$ function used by SPA decoding algorithm.

In this work, an analysis of the ADMM LDPC decoding algorithm so as to identify the performance bottlenecks is presented. Then, a novel software-optimized ADMM decoder for multi-core platforms is described. Next, a study of the complexity repartition among the decoder blocks is carried out. Through taking advantage of the architectural features of the multi-core processors and the parallelism level identified during this study, the original software implementation provided by [16] is speeded-up. The accelerated version of the ADMM algorithm can achieve up to 30 Mbps for standardized LDPC codes. Its performance features excel those of the first traditional LDPC decoders software implementations [17–21], though they are still below the recent ones on multi-core platforms [22–24]. This novel implementation paves the way for a real time software implementation. Furthermore, the accelerated ADMM decoder's performance has been simplified for study at high signal to noise ratio values.

The remainder of the paper is organized as follows: in section II, the formulation of LP decoding and its ADMM representation is reviewed. It is compared with the traditional MP approaches. In section III, a time profiling of the different blocks optimized is presented and the parallelism levels of the algorithm and their application on the target architecture to get software improvements are described. In section IV, some performance results are provided.

2. THE ADMM ALGORITHM FOR LP-LDPC DECODING

A (d_v, d_c) binary LDPC code of block length N is described by a parity check matrix $H \in \mathbb{F}_2^{M\times N}$ $(M \leq N)$ having d_v ones at each column and d_c ones at each row. It can be represented by its Tanner graph where the variable nodes (VN) are indexed by $\mathcal{I} = \{1, \ldots, N\}$ and the check nodes (CN) are indexed by $\mathcal{J} = \{1, \ldots, M\}$. The element $H_{ij} = 1$ in H means that there is an edge between the variable node v_i and the check node c_j. Let d_{v_i} be the degree of the variable node v_i and d_{c_j} be the degree of the check node c_j. Let $N_v(i) = \{j \in \mathcal{J}/H_{ij} = 1\}$ the index of the set of neighbors of variable node v_i and $N_c(j) = \{i \in \mathcal{I}/H_{ij} = 1\}$ the index of the set of neighbors of check node c_j. Once transmitted over a noisy channel, the LP decoding is equivalent to solving a linear program over a codeword polytope:

$$\begin{aligned} &\text{minimize } \gamma^T x \\ &\text{Stating that } \; Hx^T = 0, \end{aligned} \tag{1}$$

where γ is the vector of log-likelihoods.
The check polytope $P_{d_{c_j}}$ is the convex hull of all binary vectors of length d_{c_j} with an even number of ones. Barman *et al.* [14] demonstrated that the LP decoding problem (1) can be reformulated in order to fit the ADMM template given in [13]. Finally, the augmented Lagrangian of the LP problem with scaled dual variable can be written as:

$$L_\mu(x, z, \lambda) = \gamma^T x + \frac{\mu}{2} \sum_{j\in J} \|T_j x - z_j + \lambda_j\|_2^2 - \frac{\mu}{2} \sum_{j\in J} \|\lambda_j\|_2^2, \tag{2}$$

Algorithm 1 Flooding based ADMM-l_2 penalized algorithm.

Kernel 1: Initialization
for all $j \in \mathcal{J}, i \in N_c(j)$ **do**
  $(z_j^{(0)})_i = 0.5, (\lambda_j^{(0)})_i = 0, L_{ji}^{(0)} = 0.5$
end for
for all $k = 1 \rightarrow (iter\_max)$ **do**
  Kernel 2: Computation of messages $VN_i \rightarrow CN_j$
  for all $i \in \mathcal{I}, j \in N_v(i)$ **do**
    $t_i^{(k)} = \sum_{j \in N_v(i)} (L_{ji}^{(k-1)}) - \frac{\gamma_i}{\mu}$
    $L_{ij}^{(k)} = \Pi_{[0,1]^{d_v}} (\frac{1}{d_{vi} - 2\frac{\alpha}{\mu}} (t_i^{(k)} - \frac{\alpha}{\mu}))$
  end for
  Kernel 3: Computation of messages $CN_j \rightarrow VN_i$
  for all $j \in \mathcal{J}, i \in N_c(j)$ **do**
    $(z_j^{(k)})_i = \Pi_{P_{d_{c_j}}} (\rho L_{ij}^{(k)} + (1-\rho) z_j^{(k-1)} + \lambda_j^{(k-1)})$
    $(\lambda_j^{(k)})_i = \lambda_j^{(k-1)} + \rho L_{ij}^{(k)} + (1-\rho) z_j^{(k-1)} - z_j^{(k)}$
    $L_{ji}^{(k)} = (z_j^{(k)})_i - (\lambda_j^{(k)})_i$
  end for
end for
Kernel 4: Hard decision from soft-values
for all $i \in \mathcal{I}$ **do**
  $\hat{c}_i = \left[\sum_{(j \in N_v(i))} L_{ji}^{(k)} \right] > 0.5$
end for

where $\mu > 0$ is the penalty parameter, λ is the scaled dual variable.

An over-relaxation parameter ρ is added to the ADMM algorithm in order to improve its convergence [15]. Besides, it is observed that the ADMM-based LP decoder has worse error performance than the MP decoder at low SNR values [25]. To address this problem, Liu *et al.* proposed a penalized ADMM decoder which adds a penalty term into the objective function of the LP formulation. When l_2 penalty terms are used, the objective function of problem (1) is replaced with $\gamma^T x - \alpha \|x - 0.5\|_2^2$. α is a penalty parameter that can be optimized in advance.

After simplifications, the ADMM-based decoding algorithm with l_2 penalty can be expressed in the form of an iterative MP algorithm, as in Algorithm 1. Like other MP decoding algorithms, this flooding based computation works out for the check updates as well as for variable updates where all the nodes are computed simultaneously. However, the amount of exchanged messages and the VN and CN computation complexities are quite different. Similar description of the flooding based LDPC decoding for Min-Sum algorithm can be found in [24].

In Algorithm 1, $\Pi_{[0,1]^{d_v}}(a)$ is the Euclidean projection of the vector a on $[0,1]^{d_v}$ while $\Pi_{P_{d_{c_j}}}(b)$ is the Euclidean projection of the vector b on the check polytope $P_{d_{c_j}}$. ADMM LP decoding has proven its efficiency as an error correction approach [15]. Figure 1 shows the frame error rate of the ADMM decoder over an AWGN channel for two WiMAX standardized LDPC codes with different rates compared with the traditional SPA decoder. In these simulations, the maximum number of iterations for all the decoders is 200 which provides good error performance [15]. The parameters of the ADMM-l_2 penalized decoder are obtained by the parameter generator given in [26]. It can be seen that the ADMM-l_2 penalized decoder performs better than the optimal SPA decoder. For example, at FER=10^{-5}, the performance gain for the ADMM-l_2 decoder is 0.3 dB for the 1152×288 code. The performance gain for the ADMM-l_2 decoder is 0.5 dB for the 576×288 code at FER=10^{-4}.

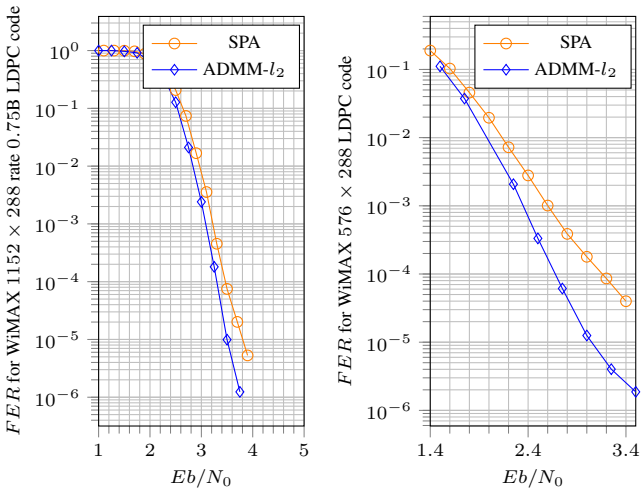

Fig. 1. FER comparison of ADMM-l_2 penalized decoders with SPA decoders on AWGN channel.

3. ALGORITHM MAPPING AND OPTIMIZATION

3.1. Target multi-core architecture

In this study, we focused mainly on developing an efficient implementation of the ADMM penalized decoding algorithm on multi-core target architectures such as x86 processor. Indeed these processors are designed essentially to support general purpose computations while providing parallel processing capabilities. Currently, x86 processor cores provide two supplementary parallel programming models: Single Instruction Multiple Data (SIMD) and Single Instruction, Multiple Threads (SIMT). These programming models enable high signal processing acceleration especially given the fact that they are associated with the fast and large memory caches available in multi-core architectures. These architectures seem to be at least as fast as GPU devices for LDPC decoding [24].

To reach the highest throughput performances, the ADMM penalized decoding algorihtm was first adapted to take advantage of both SIMD and SIMT programming models. Then, the bottleneck parts of the source code were restructured and optimized.

3.2. ADMM decoder profiling

Xishuo Liu, one of the authors of the first articles [14,25] on ADMM for LDPC codes provides his source codes online [16] under open-source license. His decoder description is not optimized for throughput performances but acts as a functional demonstrator. The work presented in this article initially took over his C++ ADMM decoder implementation. Then it was slightly modified to include the latest researches in the field [26] where the l_2 penalization function introduces different penalty parameters for variable nodes with different degrees. It improves the decoding performances of the ADMM decoder compared with the SPA approach [25].

The ADMM decoding algorithm is MP-based. When compared with the SPA decoding algorithm, the computation complexity of the VNs and CNs seem to be higher due to the multiplications and divisions in VNs and multiplications and projection in CNs. In order to identify the hot spots in terms of computation complexity, a profiling step of the application was done. Indeed, theoretical analysis is complex for the ADMM decoder where some parts of the projection

operation involved in CNs are control intensive processing.

Using the real time counter of the x86 target (RTSC instruction), we measured the average number of clock cycles required to execute kernel 2 and kernel 3 of Algorithm 1 for various SNR values. Results show that the CN kernel consumes over 80% of the execution time. Different regular and irregular LDPC codes with various code lengths as well as various check degrees $\in \{6, 7, 14, 15\}$ were simulated to provide profiling statistics. Time breakdown seems to be independent from these parameters. Moreover, additional experimentations demonstrated that the SNR value has a negligible impact on the execution time repartition.

The CN kernel (kernel 3 of Algorithm 1) is composed of two main parts. The first part prepares the values for the projection according to the VN to CN messages and updates the CN to VN messages after the projection. The computations involved are primarily additions and multiplications. The second part of the CN kernel is the Euclidean projection on the parity polytope. The projection algorithm is fully detailed in [14] section 4. This projection computation is composed of arithmetic operations interleaved with sequential control intensive sections. We carried out statistics about the execution time repartition in the CN kernel. Experimental results show that the projection consumes about 60% of the CN execution time. The complete description of the projection shows that on top of arithmetic additions and multiplications, this algorithm makes use of two sorting functions of $2d_c$ values. Profiling statistics attest that the percentage of the sorting processing cycles over the total projection time is not less than 70%.

These execution time observations show that the current bottleneck of the ADMM decoder software implementation is the projection task and therefore, it has to be optimized. Since the data sorting stages consume an important part of the projection time, a better sort function has to be identified. Meanwhile, the ADMM decoder optimization efficiency would only be reached if non-bottleneck parts of the algorithms are also optimized (Amdahl's law). In the next section we explain the different optimizations that we have performed to improve the algorithm mapping on multi-core architectures.

4. SOFTWARE OPTIMIZATIONS

4.1. Parallelism optimization

In this section, the implementation choices and the applied optimizations are explained in order to improve the ADMM software decoding speed and reach significant throughputs on multi-core devices.

First, a set of traditional optimizations was applied. An adjustment to simple precision floating point for both data and computations was performed to improve the processing efficiency without impacting on the correction performance of the decoder. Subsequently, the code was restructured and the memory footprint was minimized. The memory cost of the ADMM penalized decoder finally equals one of traditional SPA decoder ($2 \times N + 2 \times M$, with M the number of exchanged messages). Then, a revamping of the loops structures so as to bring out the flooding scheduling was done.

In addition, three typical parallelism levels were identified in Algorithm 1 through taking advantage of the general purpose multi-core processors features and the kernel computations were also manually optimized using SIMD and SIMT features. Firstly, to perform the variable nodes update computations (lines 7 to 10), each 8 VNs are processed in parallel. This gathering enables full efficiency using all 8 processor calculation units at once. However, d_v degree are often lower than 8, thus $\times 8$ acceleration can't be always achieved. Secondly, inside the CN computation (lines 13 to 15), the parallelism is applied to get the necessary vector for the projection, to perform the projection and to update the CN message. For each CN, d_c calculation units are assigned in parallel. For instance, when $d_c = 6, 6$ out of 8 calculation units are assigned, leading to 75% of efficiency. Thirdly, at the frame level computation, processing different frames in parallel is equivalent to running multiple decoders simultaneously by using a different number of threads offered by the processor. According to this analysis, we decided to take advantage of massively parallel devices by computing a set of q frames in parallel using OpenMP directives.

Finally, the set of d_c parallel computations located in the projection task was optimized using the SIMD processing capability, though a large part of the projection's processing is sequential and can not be SIMD optimized.

4.2. Sort profiling

The decoder profiling in section 3.2 underscores the need to decide how best to reduce the complexity of the sort function used in the projection algorithm. The idea is to find the fastest way to sort an array of d_c floating point values while keeping their original coordinates (for a later reordering). As a starting point, we considered and compared the sorting algorithms in the literature. First, sorting algorithms were benchmarked by sorting only 6 float values. For each algorithm, the average amount of processor clock cycles required for the sort was measured using the RTSC instruction available in the Core-i7 target. An example of measurements is shown on Figure 2 (a) which reports the average number of cycles of 5 sorting methods when d_c is equal to 6. Obviously, the direct call to the *qsort* library function and the insertion methods are the slowest, followed by the bubble sort and rank order sort. The network sorting with fast swap seems to be the most efficient method.

Then, we extended these functions considering the sort of two arrays, one for d_c floats and another for their d_c integer coordinates. Average amount of clock cycles required by modified sorting algorithm is provided in Figure 2 (b). Apparently, the rank order sort, which doesn't need any branch and stores data in registers before sorting is the most efficient.

We have used these results, among others, as a basis and decided to use the rank order approach to enhance the performance as much as possible the projection function.

5. EXPERIMENTAL RESULTS

In this section, the performance of the optimized decoder for irregular standardized LDPC codes with different lengths, rates and degree

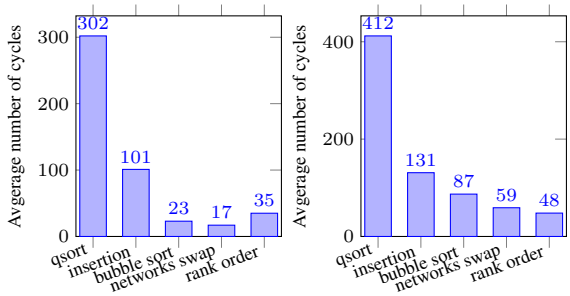

Fig. 2. Average number of cycles of (a) Reference sorting functions of 6 floats (b) Sorting functions of 6 floats keeping input positions.

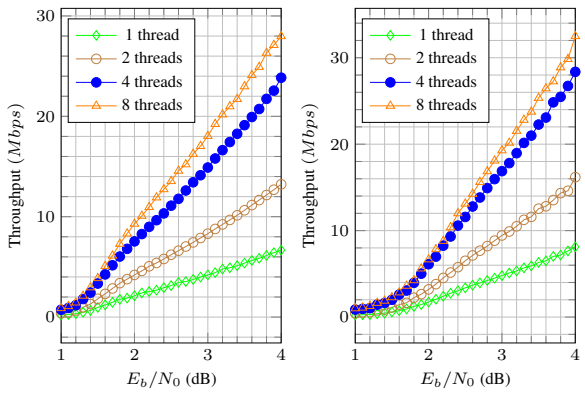

Fig. 3. ADMM-l_2 optimized decoder measured throughputs *wrt* the number of threads (a) 2304×1152 code (b) 576×288 code.

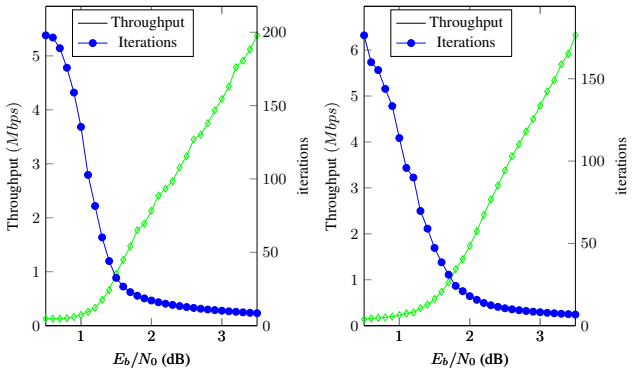

Fig. 4. Average number of iterations Vs throughput evolution (a) 2304×1152 LDPC code (b) 576×288 LDPC code.

distribution is illustrated. Then the decoding speed gain when compared with the original ADMM decoder [16] on one core processor target is shown. The evaluation platform employed is a MacBook Pro computer that runs OS X 10.10. It is composed of an INTEL Haswell Core-i74960HQ CPU. This processor runs at 2.6 GHz, with 6 MB of L3 cache memory and 16 GB of DDR3 running at 1600 MHz. It is composed of 4 Physical Cores (PC) and 4 Logical Cores (LC) sharing the L3 cache memory while each of them has 256 KB of unified L2 cache memory. Turbo-boost technology is switched on. Therefore, the processor's working clock frequency reaches 3.6 GHz when a single processor core is used and 3.4 GHz when the 4 PC and 4 LC cores are switched on.

The first experimentation set reported in Figure 3 provides the throughput performance of the multi-threaded version of the decoder when the number of processors switched on increases from 1 to 8. The throughput reaches more than 30 Mbps when 8 decoders are executed. It meets the WiMax standard requirements for the 576×288 LDPC code as well as the WRAN standard requirements for the 2304×1152 LDPC code. It is important to notice that the average number of iterations required to decode frames decreases substantially when the SNR increases. As depicted by Figure 4, the throughput increases linearly with the SNR value while the average number of decoding iterations required for the decoder to converge to a codeword decreases. For instance, at 1 dB, the average decoding iteration number reaches 114 and 135 iterations for the 576×288 and 2304×1152 LDPC codes respectively, while at 2 dB, it decreases considerably to 17 iterations.

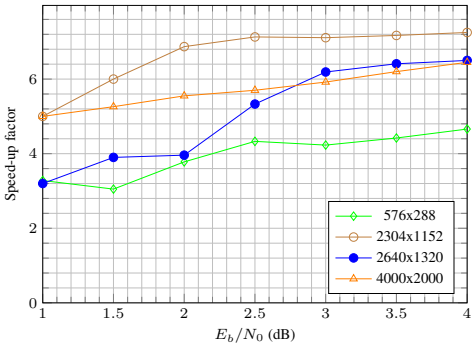

Fig. 5. Speed up factor: fast decoder throughput Vs original decoder on a single core processor for different LDPC codes.

The performance improvement in the decoding speed through comparing the throughput of the original ADMM decoder with our optimized version is shown in Figure 5. Four LDPC codes with different lengths and regularity are considered. A first observation is that the longer code length, the more important the speeding factor becomes. Another finding is that the higher the SNR value, the more important the speeding factor gets. From this figure, we can also see that there is a considerable decoding time reduction for the ultimate multi-core version. For instance, the optimized decoder runs 5 to 7 times faster than the original one starting from 2.5 dB as SNR for the long experimented codes. Lower speed-up factors are reached for irregular codes where VN and CN kernels are more complex to be optimized with the SIMD feature.

To the best of our knowledge, this study is the first implementation of an ADMM-LP decoder on multi-core architectures. The throughput performance results, though lower than those of recent LDPC decoders [22–24] (but not necessary against first implementations [17–21]), serve to strengthen the conclusion that the ADMM decoder can compete with traditional LDPC decoders on multi processors platforms, aside from having much better error correction performance. Besides, this novel optimized decoder accelerates simulations to ease the study of algorithmic simplifications at high SNR values.

6. CONCLUSION

In this paper, we described the iterative ADMM penalized algorithm used for LP decoding of LDPC codes. This LDPC decoding algorithm provides up to 0.5 dB better error correction than the MP-SPA LDPC decoder for different LDPC codes. We optimized a software implementation of the decoder to take advantage of SIMD and SIMT processing features. Optimization choices are discussed and justified according to execution profiling figures. Experimentation results show that the optimized version performs considerably better, in terms of decoding throughput, than the original version. It enables to meet Wimax standard real time throughput requirements. This achievement sheds light on a future multi processor implementation as well as on an easier evaluation of potential algorithmic simplifications to reduce the computation complexity of the ADMM LP decoding. The decoder's proposed implementation demonstrates that ADMM LDPC decoding can be a viable candidate for high correction performance in Software Defined Radio systems.

7. REFERENCES

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

[2] David JC MacKay and Radford M Neal, "Near shannon limit performance of low density parity check codes," *Electronics letters*, vol. 32, no. 18, pp. 1645–1646, 1996.

[3] "Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications, ETSI EN 302 307, 2005," .

[4] "Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications, ETSI EN 302307-2, 2014," .

[5] "IEEE 802.16-2009 Standard for Mobile WiMAX (Worldwide Interoperability for Microwave Access), Local and metropolitan area networks, Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems," .

[6] "IEEE 802.11-2012 Standard for Information technology, Telecommunications and information exchange between systems Local and metropolitan area networks, Specific requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 2012," .

[7] "IEEE 802.22-2011 Wireless Regional Area Networks (WRAN), Specific requirements, Part 22: Cognitive Wireless RAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Policies and Procedures for Operation in the TV Bands," .

[8] J. Feldman., *Decoding Error-Correcting Codes via Linear Programming*, Ph.D. thesis, Massachussets Institute of Technology, 2003.

[9] Xiaojie Zhang and Paul H.Siegel, "Efficient iterative LP decoding of ldpc codes with alternating direction method of multipliers," in *IEEE International Symposium on Information Theory (ISIT)*, 2013.

[10] J. Feldman, M.J. Wainwright, and D.R. Karger, "Using linear programming to decode binary linear codes," *IEEE Transactions on Information Theory*, vol. 51, no. 3, pp. 954–972, March 2005.

[11] Xishuo Liu, S.C. Draper, and B. Recht, "Suppressing pseudocodewords by penalizing the objective of LP decoding," in *Proceedings of the IEEE Information Theory Workshop (ITW)*, September 2012, pp. 367–371.

[12] Xiaojie Zhang and P.H. Siegel, "Adaptive cut generation algorithm for improved linear programming decoding of binary linear codes," *IEEE Transactions on Information Theory*, vol. 58, no. 10, pp. 6581–6594, June 2012.

[13] Stephen P. Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[14] Siddharth Barman, Xishuo Liu, Stark C. Draper, and Benjamin Recht, "Decomposition methods for large scale LP decoding," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 7870–7886, 2013.

[15] Xiaojie Zhang, Ph.D. thesis, University of California San Diego, 2012.

[16] "C++ ADMM Decoder by X. Liu," sites. google.com/site/xishuoliu/codes.

[17] G. Falcão, V. Silva, L. Sousa, and J. Marinho, "High coded data rate and multicodeword WiMAX LDPC decoding on the Cell/BE," *Electronics Letters*, vol. 44, no. 24, pp. 1415–1417, 2008.

[18] Juntao Zhao, Ming Zhao, Haibin Yang, Jianwen Chen, Xiang Chen, and Jing Wang, "High performance LDPC decoder on CELL BE for WiMAX system," in *Proceedings of the CMC Conference*, 2011, pp. 278–281.

[19] G. Falcao, S. Yamagiwa, V. Silva, and L. Sousa, "Parallel LDPC decoding on GPUs using a stream-based computing approach," *J. of Computer Science And Technology*, vol. 24, pp. 913–924, 2009.

[20] Y.-L. Chang and al., "High-throughput GPU-based LDPC decoding," in *Proceedings of the SPIE Satellite Data Compression, Communications and Processing conference*, 2010, vol. 7810.

[21] G. Wang, M. Wu, Y. Sun, and J. R. Cavallaro, "A massively parallel implementation of QC-LDPC decoder on GPU," in *Proceedings of the Symposium on Application Specific Processors*, 2011, pp. 82–85.

[22] Guohui Wang, Michael Wu, Bei Yin, and Joseph R. Cavallaro, "High throughput low latency LDPC decoding on GPU for SDR systems," in *Proceedings of the IEEE GlobalSIP Conference*, 2013, pp. 1258–1261.

[23] B. Le Gal, C. Jego, and J. Crenne, "A high throughput efficient approach for decoding LDPC codes onto GPU devices," *IEEE Embedded Systems Letters*, vol. 6, no. 2, pp. 29–32, 2014.

[24] B. Le Gal and C. Jego, "High-throughput multi-core ldpc decoders based on x86 processor," *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[25] Xishuo Liu and Stark C. Draper, "The ADMM penalized decoder for LDPC codes," *CoRR*, vol. abs/1409.5140, 2014.

[26] Xiaopeng Jiao, Haoyuan Wei, Jianjun Mu, and Chao Chen, "Improved ADMM penalized decoder for irregular low-density parity-check codes," *IEEE Communications Letters*, vol. 19, no. 6, pp. 913–916.