COMBINING BELIEF PROPAGATION AND SUCCESSIVE CANCELLATION LIST DECODING OF POLAR CODES ON A GPU PLATFORM

Sebastian Cammerer[*], *Benedikt Leible*[*], *Matthias Stahl*[*], *Jakob Hoydis*[‡], *Stephan ten Brink*[*]

[*]Institute of Telecommunications, Pfaffenwaldring 47, University of Stuttgart, 70569 Stuttgart, Germany
Email: {cammerer, leible, stahl, tenbrink}@inue.uni-stuttgart.de
[‡]Nokia Bell Labs, Route de Villejust, 91620 Nozay, France
Email: jakob.hoydis@nokia-bell-labs.com

ABSTRACT

The decoding performance of polar codes strongly depends on the decoding algorithm used, while also the decoder throughput and its latency mainly depend on the decoding algorithm. In this work, we implement the powerful successive cancellation list (SCL) decoder on a GPU and identify the bottlenecks of this algorithm with respect to parallel computing and its difficulties. The inherent serial decoding property of the SCL algorithm naturally limits the achievable speed-up gains on GPUs when compared to CPU implementations. In order to increase the decoding throughput, we use a hybrid decoding scheme based on the belief propagation (BP) decoder, which can be intra- and inter-frame parallelized. The proposed scheme combines excellent decoding performance and high throughput within the signal-to-noise ratio (SNR) region of interest.

I. INTRODUCTION

Polar codes are proven to be capacity achieving under successive cancellation (SC) decoding [1] for infinite block lengths. However, for short lengths, SC decoding shows a weak performance compared to state-of-the-art LDPC codes [2]. A major breakthrough in polar decoding for short length codes was achieved by Tal and Vardy with a successive cancellation list (SCL) decoder [3]. SCL decoding renders polar codes into a powerful coding scheme whenever short block lengths are required [4], as for example for the internet of things (IoT) or very low latency applications, both cornerstones of the upcoming 5G standard. However, the issue of the high SCL decoding complexity needs to be solved before polar codes can become competitive for practical applications. Besides their excellent decoding performance, the code structure of polar codes is inherently given by the concept of channel polarization [1], making them attractive for upcoming communication standards. Additionally, the code rate can be freely chosen by appropriately fixing a fraction of frozen bits.

One of the biggest current trends in the telecommunications industry is virtualization with the goal of replacing specialized hardware by software running on commodity servers [5]. However, performance critical software components, e.g., signal processing for the physical layer, require the use of hardware accelerators (GPUs, FPGAs) to satisfy the strict latency and throughput requirements of next generation communication systems. Unfortunately, not all algorithms or processing steps benefit from acceleration because they are either difficult to parallelize or the performance gains are eaten up by the overhead related to memory access. One of the most processing-resource consuming components of the physical layer is channel decoding [6]. Thus, an efficient hardware-accelerated software implementation of a decoder is of utmost importance for virtualized communication systems.

In this work we focus on graphical processing unit (GPU) implementations using the *NVIDIA Compute Unified Device Architecture (CUDA)* Framework [7]. For low-density parity-check (LDPC) codes high throughput gains were observed for LDPC belief propagation (BP) decoding [8], where massive parallelization can be done straightforwardly via parallel node updates. A similar BP algorithm can also be used for polar code decoding [9] and a correspondingly high throughput gain was observed in [10] on a GPU. Nonetheless, the implementation of the SCL algorithm for parallel computing requires more efforts, as this algorithm uses recursive updates. According to our knowledge, no GPU-based SCL decoder is proposed in the literature. Very recently an implementation of the *fast simplified successive cancellation* (fast SSC) decoder was shown in [11], but no remarkable speed-up could be observed when compared to CPU implementations[1]. Although, the overall performance is impressive, one needs to keep in mind that a quantization with 8-bit per value (32-bit floating point precision in our implementation) and no list decoding is assumed, i.e., the

[1]Remark: Several optimized SC(L) algorithms exist, mostly based on pruning/unrolling of the decoding graph. We stick to the plain SCL decoder, as this seems to be the most flexible algorithm (i.e., in terms of the code rate and variable frozen bit positions). The hybrid scheme works as well for a SSC(L) implementation of the SCL decoder. In particular for research applications, full SCL decoding is required without any further simplifications.

978-1-5090-4117-6/17/$31.00 ©2017 IEEE

decoding complexity increases (linearly) with the list size L which, typically, is set to $L = 32$.

In this paper, we identify the bottlenecks of such an implementation for parallel computing. It turns out that the SCL algorithm itself has a limited potential for GPU implementations. Therefore, we propose an alternative approach, where we combine the SCL decoder together with a belief propagation (BP) decoder. This concept combines the best of both worlds, i.e., good error correction capability and high throughputs. The authors in [12] propose a combination of SC and BP decoding and also show the possibility of using the same hardware for both algorithms. This idea is also similar to the proposed adaptive SCL decoder [13], which increases the list size whenever decoding fails. However, these algorithms increase the decoding latency, since two decoding steps are required, although the average latency improves. We examine the decoding latency for a given signal-to-noise ratio (SNR) and for different code rates.

All simulations are performed on an *Intel i7-4790K CPU @ 4.00GHz* and a *NVIDIA GTX 980 Ti* (with *6 GB GDDR5* Memory). We also provide our decoder online as a webdemo [14], where variable block lengths and arbitrary frozen bit patterns can be simulated on-the-fly (in real-time) on our servers.

II. POLAR CODES

The concept of channel polarization [1] transforms N independent channels into polarized channels by channel combining and splitting, i.e., a set of more reliable and a set of less reliable channels can be observed. The k most reliable channels are now used to transmit information bits, while the other $N - k$ bit-channels are frozen which, w.l.o.g., are set to zero. For a given code rate $R = k/N$, the k information bits are mapped onto the k non-frozen positions of $\boldsymbol{u}_N$, all other positions are frozen. The encoding process (polar transformation) can be simply described by a generator matrix $\mathbf{G} = \mathbf{F}^{\otimes n}$, where $\mathbf{F}^{\otimes n}$ denotes the n^{th} Kronecker power of $\mathbf{F} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$. The transmitted codeword is $\boldsymbol{x} = \boldsymbol{u} \cdot \mathbf{G}$. As can be seen in Fig. 1, the resulting encoding circuit has complexity $\mathcal{O}(N \cdot \log N)$.

Since the selection of the frozen positions is not the main topic of the paper, we consider a given (arbitrary) frozen bit vector $\mathbf{f}$ throughout this paper. For more details, we refer the interested reader to [15].

II-A. SC-List Decoding on GPUs

The SCL decoder utilizes the bitwise-serial decoding algorithm of the SC decoder [1] and adds a list, holding up to L of the most probable paths for the estimated codeword $\hat{\boldsymbol{x}}$ of length $N = 2^n$, resulting in a overall decoding complexity of $\mathcal{O}(L \cdot N \cdot \log(N))$ [3]. Every entry $\ell \in \{1, 2, \ldots, 2L\}$ of the list is updated for every bit decision $\hat{u}_{\ell,i}$, with i being the bit position of the estimated message bit and $\hat{\boldsymbol{u}}_\ell$ being the estimated message vector for list entry

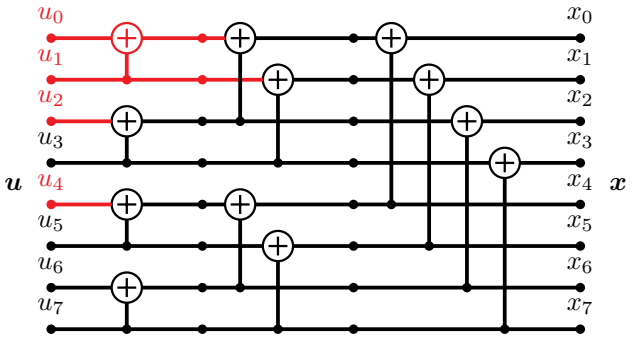


Fig. 1. Polar encoder graph for $N = 8$; red color indicates the frozen bit positions.

ℓ. List updates include sorting the branching options for each path by a metric $m_{\ell,i}$ giving the probability that the corresponding path is representing the correct estimate of the transmitted codeword. Hence, paths can get discarded and replaced by more promising candidates, which results in the need to duplicate data for newly listed branches frequently. The SCL decoding algorithm can be split up into two parts for each decision $\hat{u}_{\ell,i}$ that is made concerning an unfrozen bit. Log-likelihood ratios (LLR) are given by $LLR(y_j) = \ln\left(\frac{P(y_j|x_j=0)}{P(y_j|x_j=1)}\right)$, where $P(y_j|x_j)$ denotes the conditional probability that the channel output y_j is received while the codeword element x_j was transmitted. First an $LLR(\hat{u}_{i,\ell}) = \ln\left(\frac{P(\boldsymbol{y}, \hat{\boldsymbol{u}}_{0,\ell}^{i-1}|u_i=0)}{P(\boldsymbol{y}, \hat{\boldsymbol{u}}_{0,\ell}^{i-1}|u_i=1)}\right)$ is calculated from the received codewords $\boldsymbol{y}$ and the previously decided bits $\hat{\boldsymbol{u}}_{0,\ell}^{i-1}$ of the respective list entry ℓ. Subsequently, the calculated values are used to decide which of the maximal $2L$ branching options are kept in the list for the next decoding phase. Additionally, a cyclic redundancy check (CRC) can be added, aiding the selection of the estimated codeword after the last decision step [3]. This further improves the decoding capabilities of the SCL decoder and results in a negligibly smaller code rate $R_{\text{crc}} = \frac{k-c}{N}$, with c being the number of CRC parity bits.

For the implementation, the CUDA framework by NVIDIA is used, which enables the use of commodity graphics hardware for single instruction multiple data (SIMD) parallel programming [7]. Due to high data dependencies between the bitwise estimation steps, the algorithm of the SCL decoder is mostly serial by design. There seems to be no straightforward approach for a massively parallel implementation of this particular decoder, which results in impractical throughputs for a parallel implementation of the standard SCL decoder. Even though the bitwise decoding steps cannot be parallelized due to data dependencies, many of the calculations per step can be parallelized. Also, multiple list places can be processed in parallel but have to be synchronized for every decision on an unfrozen bit.

For sorting the $2L$ candidate paths by their probabilities, a parallel bitonic sort [16] was implemented. Additionally, we

propose a "pruned pseudo sort" to determine the L smallest path metrics in parallel. This pseudo sort algorithm calculates $d_{\ell,i} = \sum_{j=1}^{2L} \mathrm{H}(m_{\ell,i} - m_{j,i})$ for all $\ell \in \{1, 2, \ldots, 2L\}$, with $\mathrm{H}(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$ being the Heaviside function, and discards all candidates with $d_{\ell,i} > L$.

Further, the *decision aiding mechanism*, described in [17], was implemented to reduce the cases in which the decoder has to execute the sort and duplicate methods, which represent the dominant bottleneck reducing the achievable throughput. Instead, the more reliable (information) bit positions are estimated without branching the list (according to their Bhattacharya parameter [1]). For a carefully selected amount of bits, the decoding performance does not degrade, while the throughput strongly increases [17].

II-B. Belief Propagation Decoding of Polar Codes

BP decoding of polar codes is a message passing algorithm based on the encoding scheme shown in Fig. 1 with decoding complexity $\mathcal{O}(n \cdot \log n)$. The transmitted codeword $\hat{\boldsymbol{x}}$ and the message $\hat{\boldsymbol{u}}$ can be both estimated simultaneously. There are $n+1$ stages with N nodes per stage. Each stage consists of $N/2$ processing elements (PEs) which are iteratively passing messages to adjacent nodes. Each PE connects 4 nodes in 2 consecutive stages as shown in Fig. 2, the input/output relation of the PEs is the same as in Fig. 1. All messages are calculated in the LLR domain. One decoding iteration consists of two steps, where the soft messages are updated at each PE (until reaching a maximum number of iterations) as follows:

1) Update left-to-right messages, called $\mathbf{R}$-messages, for stages $2, ..., n+1$

$$R_{\mathrm{out},1} = \mathrm{g}(R_{\mathrm{in},1}, L_{\mathrm{in},2} + R_{\mathrm{in},2})$$
$$R_{\mathrm{out},2} = \mathrm{g}(R_{\mathrm{in},1}, L_{\mathrm{in},1}) + R_{\mathrm{in},2}$$

2) Update right-to-left messages, called $\mathbf{L}$-messages, for stages $n, ..., 1$

$$L_{\mathrm{out},1} = \mathrm{g}(L_{\mathrm{in},1}, L_{\mathrm{in},2} + R_{\mathrm{in},2})$$
$$L_{\mathrm{out},2} = \mathrm{g}(R_{\mathrm{in},1}, L_{\mathrm{in},1}) + L_{\mathrm{in},2}$$

where $\mathrm{g}(a,b) = \ln\left(\frac{1+\mathrm{e}^{a+b}}{\mathrm{e}^{a}+\mathrm{e}^{b}}\right)$. For the $\mathrm{g}(\cdot)$-function, a min-approximation $\mathrm{g}(a,b) = \mathrm{sign}(a) \cdot \mathrm{sign}(b) \cdot \min(|a|, |b|)$ can be used which is more suitable for hardware implementation [18]. An advantage of GPU computing is the availability of many floating-point units (FPU). Thus, we apply the exact node update-equations with clipping the absolute values of the LLR values to $LLR_{max} = 20$ to ensure numerical stability.

We initialize $L_{i,n+1} = LLR(y_i)$ and $R_{i,1} = LLR_{max} \cdot f_i$. The final output of the decoder is

$$LLR(\hat{u}_i) = L_{i,1} + R_{i,1}$$
$$LLR(\hat{x}_i) = L_{i,n+1} + R_{i,n+1}.$$

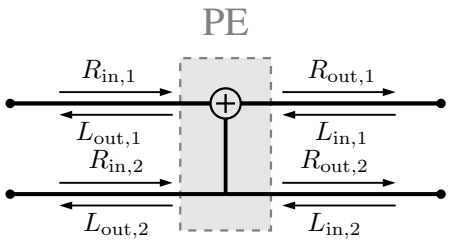


Fig. 2. Single processing element (PE) of the BP decoder.

A hard-decision gives the estimated bit-vectors, while for other applications such as a combined channel-detection, soft-values may be required and can be easily forwarded. Additionally, stopping conditions [19] exist, and thus, the decoding can be done within a few iterations for most of the frames (in the high SNR region) as soon as $\hat{\boldsymbol{x}} = \hat{\boldsymbol{u}} \cdot \mathbf{G}$. For this specific setup, a simple CRC on $\hat{\boldsymbol{u}}$ can replace the more complex re-encoding step, i.e., decoding is stopped whenever the CRC of $\hat{\boldsymbol{u}}$ is fulfilled (or a maximum of iterations is performed).

We apply intra- and inter-frame parallelism, i.e., several codewords are decoded in parallel, where each single thread implements one PE. Nevertheless, synchronization after each stage is required.

III. COMBINING BP AND SCL DECODING

The bit-error-rate (BER) performance of polar codes under SCL decoding is better than that under BP decoding [4]. However, in terms of suitability of parallelization, the BP decoder shows a higher potential because all bits can be calculated in parallel while the latency can be decreased. This is shown in Fig. 3 and Fig. 5, respectively.

Whenever the CRC after the maximum number of BP iterations $i_{\max}$ does not hold, the codeword is forwarded to an additional SCL decoding step. It turns out that only $i_{\max} = 50$ BP iterations are sufficient, otherwise each decoding failure blocks GPU ressources for a long time.[2]

SCL decoding could be performed on the CPU as well, however, the required data transfer produces additional latency overhead and is thus not advisable.

IV. DECODING PERFORMANCE

For a given BP frame-error-rate (FER) $\gamma_{BP,FER}$ and a (information bit) throughput T_{BP} and T_{SCL} for the BP and SCL decoder, respectively, the overall throughput can be approximated (without kernel-call overhead) as

$$T_{hyb,theo} = \frac{T_{BP} \cdot T_{SCL}}{T_{SCL} + \gamma_{BP,FER} \cdot T_{BP}}. \quad (1)$$

The measured throughput is depicted in Fig. 4, the gap between theoretical and measured throughput can be explained

[2]Remark: The overall BER performance does not (significantly) depend on $i_{\max}$, since we assume that SCL decoding can decode (practically) all noisy codewords anyway, which could be decoded under BP decoding.

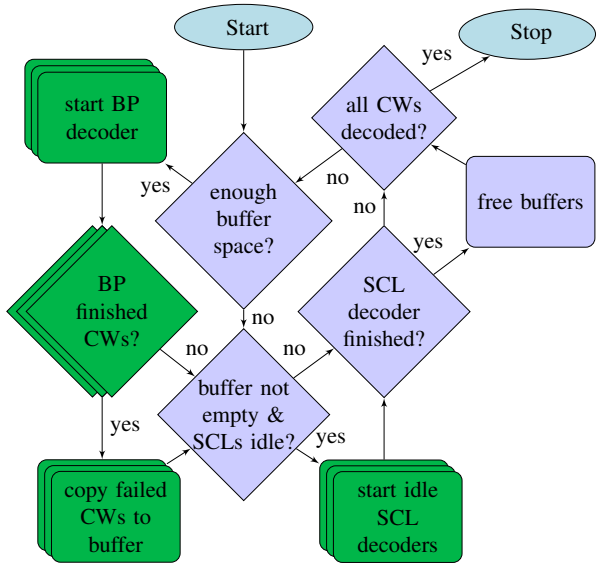


Fig. 3. Scheduling mechanism of the hybrid decoder (grey nodes: CPU, green nodes: GPU).

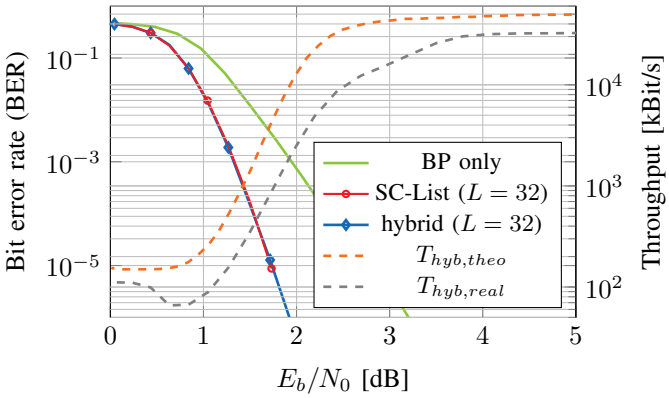
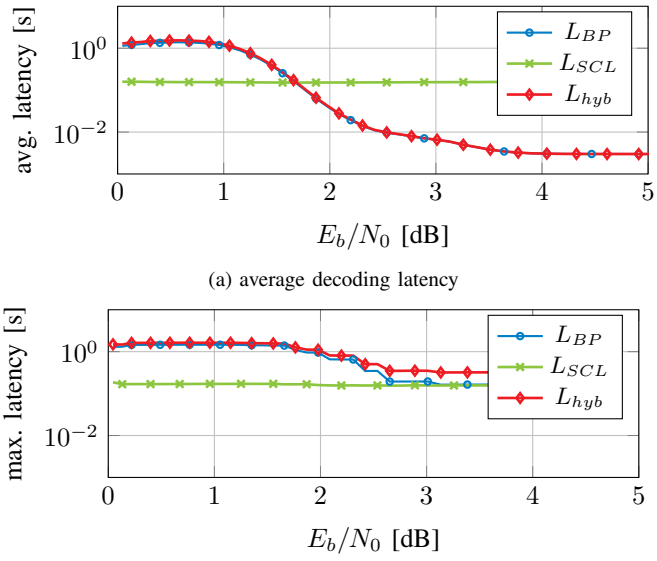


Fig. 4. Hybrid decoder throughput $T_{hyb,real}$ measurement, approximated throughput $T_{hyb,theo}$ and BER performance of the BP, the SCL and the hybrid decoder with $N = 4096$ and $R = 0.5$.

by the kernel-call overhead, which is not considered in (1). A maximum decoding throughput of 34 Mbit/s can be achieved for $N = 4096$, $L = 32$ and $R = 0.5$. Additionally, it can be seen that the BER does not differ from the SCL curve.

We need to emphasize that the operation point of channel codes is typically not in the high BER region (as no reliable communication is possible) and thus high speed-up gains are observed in practice. At least for low (and intermediate) BER applications, e.g., error-floor analysis which requires a lot of simulation time, this proposal shows a huge improvement by a factor of 100 and more, when compared to the non-hybrid implementation of the SCL decoder.



(a) average decoding latency

(b) worst-case decoding latency

Fig. 5. Decoding latency of the BP decoder L_{BP}, the SCL decoder L_{SCL} and the hybrid decoder L_{hyb}. Code parameters are $N = 4096$, $L = 32$, $R = 0.5$ and $i_{max} = 50$.

IV-A. Latency

Whenever BP decoding fails, SCL decoding must be performed; then, the total latency L_{hyb} increases as shown in Fig. 5. As the BP frame-error-rate is $\gamma_{BP,FER} \ll 1$, the average latency mainly depends on L_{BP}. The BP decoding latency is strongly related to the SNR due to the implemented early stopping mechanism. In the low SNR region it is even above the SCL latency, which stems from the fact that multiple BP codewords are decoded in parallel, i.e., computation resources need to be shared. However, when compared to SCL decoding, the average latency only increases slightly since $L_{BP} \ll L_{SCL}$ for high SNR, i.e., only a few applied BP iterations. When compared to other adaptive decoding concepts such as e.g., an adaptive list size [13], our approach shows better latency performance in the target SNR region.

V. CONCLUSIONS

In this work, we present a first SCL *CUDA* implementation for GPU computing and evaluate the achievable decoding throughputs. As this algorithm turns out to be very challenging to speed-up, a combination with BP decoding is considered. It is shown, that the BP algorithm can be easily adapted for GPU computing. The decoding throughput of the novel hybrid approach can be drastically increased, while the average decoding latency decreases when compared to SCL decoding. Further, in contrast to LDPC codes, almost no publications regarding GPU-based polar decoding exist and we hope that this work inspires other research groups to further investigate this challenging problem.

VI. REFERENCES

[1] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.

[2] E. Arıkan, N. ul Hassan, M. Lentmaier, G. Montorsi, and J. Sayir, "Challenges and some new directions in channel coding," *J. Commun. Netw.*, vol. 17, no. 4, pp. 328–338, Aug. 2015.

[3] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015.

[4] K. Niu, K. Chen, J. Lin, and Q. T. Zhang, "Polar codes: Primary concepts and practical decoding algorithms," *IEEE Commun. Mag.*, vol. 52, no. 7, pp. 192–203, Jul. 2014.

[5] P. Rost, I. Berberana, A. Maeder, H. Paul, V. Suryaprakash, M. Valenti, D. Wübben, A. Dekorsy, and G. Fettweis, "Benefits and challenges of virtualization in 5G radio access networks," *IEEE Commun. Mag.*, vol. 53, no. 12, pp. 75–82, 2015.

[6] N. Nikaein, "Processing radio access network functions in the cloud: Critical issues and modeling," in *Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services*. ACM, 2015, pp. 36–43.

[7] NVIDIA, "Performance guidelines," *CUDA C Programming Guide*, Aug. 2014.

[8] G. Falcao, L. Sousa, and V. Silva, "Massively LDPC decoding on multicore architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 2, pp. 309–322, Feb. 2011.

[9] E. Arıkan, "A performance comparison of polar codes and reed-muller codes," *IEEE Comm. Letters*, vol. 12, no. 6, Jun. 2008.

[10] B. K. Reddy L. and N. Chandrachoodan, "A GPU implementation of belief propagation decoder for polar codes," in *Proc. Asilomar Conf. on Signals, Systems, and Computers*, Nov. 2012, pp. 1272–1276.

[11] P. Giard, G. Sarkis, C. Leroux, C. Thibeault, and W. J. Gross, "Low-latency software polar decoders," *CoRR*, vol. abs/1504.00353, 2015. [Online]. Available: http://arxiv.org/abs/1504.00353

[12] B. Yuan and K. K. Parhi, "Algorithm and architecture for hybrid decoding of polar codes," in *Proc. Asilomar Conf. on Signals, Systems, and Computers*, Nov. 2014, pp. 2050–2053.

[13] B. Li, H. Shen, and D. Tse, "An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check," *IEEE Comm. Letters*, vol. 16, no. 12, pp. 2044–2047, Dec. 2012.

[14] B. Leible, M. Stahl, and S. Cammerer, "On-the-fly Polar Code Bit-Error-Rate Simulations," Institute of Telecommunications, University of Stuttgart, Germany, Oct. 2016, http://webdemo.inue.uni-stuttgart.de.

[15] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6562–6582, Oct. 2013.

[16] H. Peters, O. Schulz-Hildebrandt, and N. Luttenberger, "Fast in-place, comparison-based sorting with CUDA: A study with bitonic sort," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 7, pp. 681–693, 2011.

[17] B. Li, H. Shen, and K. Chen, "A decision-aided parallel SC-list decoder for polar codes," *CoRR*, vol. abs/1506.02955, 2015. [Online]. Available: http://arxiv.org/abs/1506.02955

[18] Y. S. Park, Y. Tao, S. Sun, and Z. Zhang, "A 4.68Gb/s belief propagation polar decoder with bit-splitting register file," in *Proc. IEEE Int. Symp. on VLSI*, Jun. 2014, pp. 1–2.

[19] B. Yuan and K. K. Parhi, "Early stopping criteria for energy-efficient low-latency belief-propagation polar code decoders," *IEEE Trans. Signal Process.*, vol. 62, no. 24, pp. 6496–6506, Dec. 2014.