2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME) | 979-8-3503-2297-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICECCME57830.2023.10253123

Proc. of the International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME 2023)
19-21 July 2023, Tenerife, Canary Islands, Spain

Software Implementation and Performance of a Computational Complexity Reduced Belief Propagation Polar Code Decoder

Arvid B. van den Brink
Department of Computer Architectures for Embedded Systems University of Twente,
Enschede, The Netherlands
a.b.vandenbrink@utwente.nl

Oğuz Meteer
Department of Computer Architectures for Embedded Systems University of Twente,
Enschede, The Netherlands
o.meteer@utwente.nl

Marco J.G. Bekooij
Department of Computer Architectures for Embedded Systems University of Twente,
Enschede, The Netherlands
Department of Embedded Software and Signal Processing NXP Semiconductors,
Eindhoven, The Netherlands
marco.bekooij@nxp.com

***Abstract*—The belief propagation algorithm is one of the preferred algorithms for a polar code based decoder that can potentially achieve the lowest latency. Although belief propagation polar code algorithms have the ability for a highly parallelized implementation, they require more iterations to achieve a comparable frame error rate and bit error rate compared to the successive cancellation polar code algorithm. The iterative nature of the belief propagation algorithms also results in a higher computational complexity, i.e. O(N(2log$_2$N-1)) compared to the computational complexity O(Nlog$_2$N) of the successive cancellation decoder algorithm. In this paper we propose several software implementations of belief propagation polar code decoders using the computational complexity reduced belief propagation algorithm and the enhanced loop-weakened belief propagation algorithm, to increase the number of decoded codes per second. Compared to a baseline belief propagation implementation, our proposed radix-2 and radix-4 implementations increase the throughput by approximately 15.4% and 22.04% respectively. This gain is the result of a reduction of arithmetic operations, i.e., additions, compares, and multiplications, which is obtained without a loss in error-correcting performance.**

***Index Terms*—Complexity, Belief Propagation, Polar Code, Algorithm**

I. Introduction

Wide spread use of codes that approach capacity, such as low-density parity check (LDPC) [2] and Turbo Codes [1] can be seen in applications such as data storage and wireless communication, and the first provably capacity-achieving code called polar code was introduced by Arikan [3]. Its provably capacity-achieving error correction capability for any binary-input discrete memoryless channel (B-DMC) [4] has given this type of codes much attention in recent years. The industry's attention grew, when the 3^{rd} generation partnership project (3GPP) [5] had chosen polar codes as one of the encoding schemes for the downlink and uplink control channels in 5^{th} generation wireless systems (5G).

For decoding polar codes, the low complexity decoding algorithm successive cancellation (SC) was proposed, which has a computational complexity $O(N \log N)$, where N is the length of the codeword. With an increasing block length N, the decoding latency and computational complexity is becoming a bottleneck, therefore shorter block lengths are preferred, especially for applications which are latency critical, like vehicle-to-X communication (V2X), or applications with constrained resources such as Internet of Things (IoT) devices.

In this paper, we focus on potential software implementation improvements of the BP decoding algorithm in terms of arithmetic complexity. In a similar fashion, the authors of [8] have explored hardware implementation improvements. By using simplified equations in the BP algorithm [9], a new algorithm is obtained with a lower arithmetic complexity, i.e. with less operations like additions and comparisons. The simplified equations will use the enhanced loop-weakened method as described in [10]. We present a radix-2, and radix-4 software BP polar code decoder implementation that, compared to the textbook BP algorithm for a $\mathcal{P}(1024, 512)$ polar code, has an increase in the branch miss rate (BMR), yet decreases the execution time and thus increases throughput in number of decoded codes per second. Additionally, the FER and BER of the proposed algorithm is not changed compared to the BP algorithm.

The remainder of this paper is composed as follows. Section II gives an overview of related work. In Section III a general review of polar codes is given, and in particular the BP decoding algorithm. Our proposed software implementation of the computational complexity reduced BP decoding algorithm is presented in Section V. In Section VI an evaluation of the baseline and proposed software implementations is given. Finally, in Section VII we state the conclusions.

II. Related Work

Since the introduction of polar codes as a capacity achieving code for any binary-input discrete memoryless channel (B-DMC) [3], polar codes have garnered much interest in the research community [14]–[16]. Although the proven capacity achieving property is for infinite code length SC decoding,

979-8-3503-2297-2/23/$31.00 © 2023 IEEE

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on March 13,2024 at 14:44:51 UTC from IEEE Xplore. Restrictions apply.

the error correcting performance is not assured for finite length polar codes. The sequential decoding structure of the SC algorithm is another hurdle to take, as it results in a lower throughput for longer code lengths. The successive cancellation list decoding algorithm [15] was proposed, to reduce the error correcting performance degradation of the successive cancellation algorithm. It explores multiple paths of the decoding tree and chooses the best candidate as the decoder output. This will increase the error correcting performance of the decoder, but consequently also increases the computational complexity of the decoder.

To increase the throughput of a polar code decoder, the BP algorithm [11] can be used, which allows a highly parallel execution. To improve the error correcting performance, a list variant of the BP decoder is proposed in [17]. To reduce the number of iterations, several early stopping criteria are proposed in [18]–[20]. For increasing the throughput, several architectural optimizations are proposed in [21]. Reducing the memory requirement of the BP decoder is proposed in [22]. The number of memory operations are reduced by a stage combined BP decoder as proposed in [13]. A vectorized BP decoder is proposed in [23], to address the memory bottleneck. A reduced computational complexity BP algorithm is proposed in [9]. An increase in error correction rate as well as a decrease in latency is proposed in [10]. Finally, a hardware-centric version of the proposed software solution in this paper was implemented and evaluated in [8].

III. Preliminary

A. Polar codes

Polar codes, with abbreviated parameter vector $\mathcal{P}(N, K)$, on any symmetric binary-input discrete memoryless channel (B-DMC), like the binary symmetric channel (BSC), are designed as a linear, and capacity achieving block code. Using the definition in [3], they are represented by a parameter vector $(N, K, \mathcal{A}, u_{\mathcal{A}^c})$ with $N = 2^n \ \forall n > 1$, with codeword length N, K information bits, the set of information bits $\mathcal{A}$, and $u_{\mathcal{A}^c}$ represents the set of predetermined (frozen) bit values, called frozen bits. The encoding and decoding of the source vector of a polar code consists of the following steps:

1) The source vector $u_1^N = (u_1, \cdots, u_N)$ is encoded using:
$$x_1^N = u_1^N G_N = u_1^N B_N F^{\otimes n}, \tag{1}$$
with generator matrix G_N, permutation matrix B_N, also known as the bit-reversal matrix, and $F^{\otimes n}$ is the *Kronecker* power of F defined as:
$$F^{\otimes n} = F \otimes F^{\otimes (n-1)}, \tag{2}$$
with $n = \log_2 N$, and $F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$.
2) x_1^N is sent using the constructed virtual channels W^N of the polar code, feeding the channel output $y_1^N = (y_1, y_2, \cdots, y_N)$ into the decoder.
3) An estimation $\hat{u}_1^N$ is made using the polar codes $\mathcal{A}$, $u_{\mathcal{A}^c}$ and output of the channely_1^N.

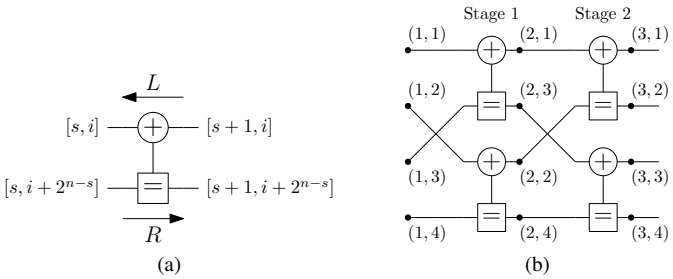

Fig. 1. (a) computational unit used in Belief propagation algorithm; (b) Length $N = 4$ factor graph used for polar codes.

B. Belief propagation polar decoding

A belief propagation polar code decoder is created, using the factor graph based BP polar code algorithm, as described in [11]. A 2-bits factor graph of a radix-2 BP polar code decoder is shown in Fig. 1a. A 4-bits polar code decoder, where each stage is constructed with $N/2$ computational units (CUs), is shown in Fig. 1b. Every CU consist of four terminals, as shown in Fig. 1a. The terminals are mapped to:

$$L_{s,i}^{(m+1)} = h\big(L_{s+1,i}^{(m)}, L_{s+1,i+2^{n-s}}^{(m)} + R_{s,i+2^{n-s}}^{(m)}\big), \tag{3}$$
$$L_{s,i+2^{n-s}}^{(m+1)} = h\big(R_{s,i}^{(m)}, L_{s+1,i}^{(m)}\big) + L_{s+1,i+2^{n-s}}^{(m)}, \tag{4}$$
$$R_{s+1,i}^{(m+1)} = h\big(R_{s,i}^{(m)}, L_{s+1,i+2^{n-s}}^{(m)} + R_{s,i+2^{n-s}}^{(m)}\big), \tag{5}$$
$$R_{s+1,i+2^{n-s}}^{(m+1)} = h\big(R_{s,i}^{(m)}, L_{s+1,i}^{(m)}\big) + R_{s,i+2^{n-s}}^{(m)}, \tag{6}$$

with $1 \leq m \leq M$, where M is the maximum number of iterations, and the approximation of the min-sum function h is:

$$h(v, w) \approx 0.9 \cdot \varphi(v) \cdot \varphi(w) \cdot \min(|v|, |w|), \tag{7}$$

with $\varphi(x)$ returns the sign of x.

The iterative process of the BP algorithm, uses the constructed factor graph to process messages. Two message types are involved in decoding: left bound messages L and right bound messages R. Each node' message in the factor graph is assigned an initial log likelihood ratio (LLR) value, depending on its index.

The R messages on the left are assigned a value using (8).

$$R_{1,i} = \begin{cases} 0, & \text{if } j \in \mathcal{A}; \\ \infty, & \text{if } j \in \mathcal{A}^c, \end{cases} \tag{8}$$

with $\mathcal{A}^c$ the set of predetermined bits and $\mathcal{A}$ the information bits set.

The L messages on the right are assigned an LLR using (9).

$$L_{n+1,i} = \ln\left(\frac{\Pr[y_i \mid x_i = 0]}{\Pr[y_i \mid x_i = 1]}\right). \tag{9}$$

All other node messages are initially set to zero.

After M iterations, the decoder's output $\hat{u}_1^N$ is estimated using threshold detection at the left most terminals using (10).

$$\hat{u}_i = \begin{cases} 0, & \text{if } L_{1,i} + R_{1,i} \geq 0; \\ 1, & \text{otherwise.} \end{cases} \tag{10}$$

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on March 13,2024 at 14:44:51 UTC from IEEE Xplore. Restrictions apply.

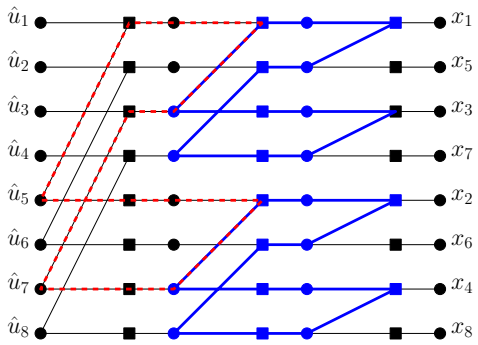

Fig. 2. Several size 12 cycles in a $N = 8$ polar code Tanner graph, indicating the girth of this graph. Check nodes are shown as squares, variable nodes as circles.

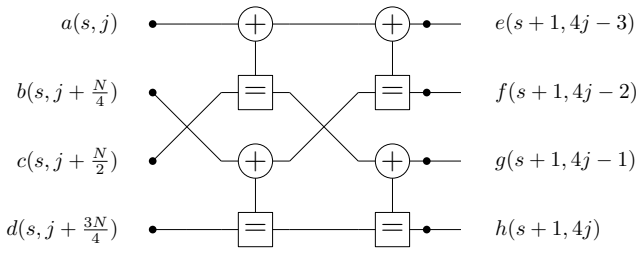

Fig. 3. A length $N = 4$ Radix-4 polar code factor graph.

C. Girth of a graph

The girth of a graph is the length of the shortest cycle. In an iterative algorithm like BP, up to the point where the iteration reaches the graph's girth, every variable node's information remains uncorrelated. The higher the girth of the BP algorithm is, the better decoding performance the code has. In [12] (and references in the publication), the proposed technique is to guarantee a girth equal or larger then 12, as this girth is considered desirable for any code constructions, like polar codes.

In Fig.2, some of the shortest cycles in a polar code of $N = 8$ are shown, which are of length 12 in this case. Due to the recursive nature of the code construction, a polar code of larger code length will also have a girth of 12.

D. Stage-Combined Belief Propagation Decoding

While the standard BP algorithm is using CUs of type radix-2, a radix-4, stage-combined BP decoder was proposed in [13]. The proposed radix-4 BP algorithm has two benefits, without any performance degradation. First, decoder latency is reduced by reducing the number of stages in the graph from $2\log_2(N) - 1$ to $\log_2(N) - 1$. Second, a radix-4 decoder is more efficient at decoding larger codewords than a radix-2 decoder because the number of messages/operation is reduced from $N\log_2(N) - 1$ to $N\log_4(N) - 1$.

In Fig.3 the proposed radix-4 factor graph is shown. The terminals a through h are mapped to:

$$L_a = h(L_e, h(L_g, R_b) + h(R_c, L_f) + h(L_h + R_d, h(R_c, R_b) + h(L_g, L_f))), \quad (11)$$

$$L_b = h(L_f, h(L_g, R_c) + L_h + R_d) + h(L_e, h(R_a, L_g + h(L_h + R_d, R_c))), \quad (12)$$

$$L_c = h(L_g, h(L_f, R_b) + L_h + R_d) + h(L_e, h(R_a, L_f + h(R_b, L_h + R_d))), \quad (13)$$

$$L_d = L_h + h(L_g, R_c) + h(L_f, R_b) + h(L_e, h(R_a, h(R_c, R_b) + h(L_g, L_f))), \quad (14)$$

$$R_e = h(R_a, h(L_g, R_b) + h(R_c, L_f) + h(L_h + R_d, h(R_c, R_b) + h(L_g, L_f))), \quad (15)$$

$$R_f = h(R_b, h(L_g, R_c) + L_h + R_d) + h(R_a, h(L_e, R_c + h(L_h + R_d, L_g))), \quad (16)$$

$$R_g = h(R_c, h(L_f, R_b) + L_h + R_d) + h(R_a, h(L_e, R_b + h(L_f, L_h + R_d))), \quad (17)$$

$$R_h = R_d + h(L_g, R_c) + h(L_f, R_b) + h(R_a, h(L_e, h(R_c, R_b) + h(L_g, L_f))). \quad (18)$$

IV. Basic Idea

The output's LLRs of the nodes in a polar code factor graph are dictated by their input values and their position in the factor graph [9], where the cycle-12 count is reduced. The BP algorithm can also converge faster, hence reducing the number of required iterations as described in [10].

In our proposal we implemented these proposed enhancements for both radix-2 and radix-4 BP algorithm on an ARM Cortex R4 processor.

V. Proposed Belief Propagation Algorithms

As a baseline, we created a radix-2 BP C implementation, using (3)-(6), and an equal baseline C implementation for a radix4 BP, using the (11)-(18), which are described in section V-A.

For the comparison, we then implemented the computational complexity reduced BP algorithm [9], using the enhanced loop-weakened BP algorithm [10], for both the radix-2 and radix-4 polar code decoders, which is described in section V-B.

A. Radix-2 and radix-4 baseline implementations

The baseline implementation of the radix-2 BP decoder is described in Alg. 1, and the baseline implementation of the radix-4 BP decoder is described in Alg. 2, where in both algorithms N is the code word length, and $n = \log_2(N)$. The ComputeLLR function is equal to (3)-(6) and (11)-(18) for radix-2 and radix-4 respectively, where the scaling factor of 0.9 in (7) is replaced by 0.9375. This value is often used in hardware implementations and does not affect the decoding performance in any significant way.

Algorithm 1: Baseline radix-2BP decoder

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on March 13,2024 at 14:44:51 UTC from IEEE Xplore. Restrictions apply.

Initialize $\mathbf{Mem}$ ▷ Message memory
while not $Iteration_{max}$ **do**
  for $Stage \in 0, \cdots, n-1$ **do** ▷ Compute right bound
    for $CU \in 0, \cdots, \frac{N}{2}-1$ **do**
      $R_a, R_b \leftarrow Mem(CU, Stage)$
      $L_c, L_d \leftarrow Mem(CU, Stage+1)$
      $R_c \leftarrow$ ComputeLLR(R_a, R_b, L_d) ▷ (5)
      $R_d \leftarrow$ ComputeLLR(R_a, R_b, L_c) ▷ (6)
      $Mem(CU, Stage+1) \leftarrow R_c, R_d$
    end for
  end for
  for $Stage \in n-1, \cdots, 1$ **do** ▷ Compute left bound
    for $CU \in 0, \cdots, \frac{N}{2}-1$ **do**
      $R_a, R_b \leftarrow Mem(CU, Stage)$
      $L_c, L_d \leftarrow Mem(CU, Stage+1)$
      $L_a \leftarrow$ ComputeLLR(R_b, L_c, L_d) ▷ (3)
      $L_b \leftarrow$ ComputeLLR(R_a, L_c, L_d) ▷ (4)
      $Mem(CU, Stage) \leftarrow L_a, L_b$
    end for
  end for
  for $CU \in 0, \cdots, \frac{N}{2}-1$ **do** ▷ Compute final stage
    $R_a, R_b \leftarrow Mem(CU, Stage=0)$
    $L_c, L_d \leftarrow Mem(CU, Stage=1)$
    $L_a \leftarrow$ ComputeLLR(R_b, L_c, L_d) ▷ (3)
    $L_b \leftarrow$ ComputeLLR(R_a, L_c, L_d) ▷ (4)
    $Mem(CU, Stage=0) \leftarrow L_a, L_b$
  end for
end while

Algorithm 2: Baseline radix-4BP decoder

Initialize $\mathbf{Mem}$ ▷ Message memory
while not $Iteration_{max}$ **do**
  for $Stage \in 0, \cdots, \frac{n}{2}-1$ **do** ▷ Compute right bound
    for $CU \in 0, \cdots, \frac{N}{4}-1$ **do**
      $R_a, R_b, R_c, R_d \leftarrow Mem(CU, Stage)$
      $L_e, L_f, L_g, L_h \leftarrow Mem(CU, Stage+1)$
      $R_e \leftarrow$ ComputeLLR(R_*, L_*) ▷ (15)
      $R_f \leftarrow$ ComputeLLR(R_*, L_*) ▷ (16)
      $R_g \leftarrow$ ComputeLLR(R_*, L_*) ▷ (17)
      $R_h \leftarrow$ ComputeLLR(R_*, L_*) ▷ (18)
      $Mem(CU, Stage+1) \leftarrow R_e, R_f, R_g, R_h$
    end for
  end for
  for $Stage \in \frac{n}{2}-1, \cdots, 1$ **do** ▷ Compute left bound
    for $CU \in 0, \cdots, \frac{N}{4}-1$ **do**
      $R_a, R_b \leftarrow Mem(CU, Stage)$
      $L_c, L_d \leftarrow Mem(CU, Stage+1)$
      $L_a \leftarrow$ ComputeLLR(R_*, L_*) ▷ (11)
      $L_b \leftarrow$ ComputeLLR(R_*, L_*) ▷ (12)
      $L_c \leftarrow$ ComputeLLR(R_*, L_*) ▷ (13)
      $L_d \leftarrow$ ComputeLLR(R_*, L_*) ▷ (14)
      $Mem(CU, Stage) \leftarrow L_a, L_b, L_c, L_d$
    end for
  end for
  for $CU \in 0, \cdots, \frac{N}{4}-1$ **do**
    $R_a, R_b \leftarrow Mem(CU, Stage=0)$
    $L_c, L_d \leftarrow Mem(CU, Stage=1)$
    $L_a \leftarrow$ ComputeLLR(R_*, L_*) ▷ (11)
    $L_b \leftarrow$ ComputeLLR(R_*, L_*) ▷ (12)
    $L_c \leftarrow$ ComputeLLR(R_*, L_*) ▷ (13)
    $L_d \leftarrow$ ComputeLLR(R_*, L_*) ▷ (14)
    $Mem(CU, Stage=0) \leftarrow L_a, L_b, L_c, L_d$
  end for
end while

B. Proposed radix-2 and radix-4 implementations

In [9], it was shown that while performing BP decoding, the input LLRs with the values 0 and $+\mathbb{U}_{max}$ (predefined maximum LLR value) appear in combinations that result in simplified decoding equations, which consist of addition and the h-function of (7). If one of the variables of the h operation is equal to 0, the result is also 0, so this operation can be dropped. Since adding 0 does not change the result, the same applies to this operation. If, however, one of the inputs of the h or addition operation is equal to $+\mathbb{U}_{max}$, then only the h operation can be partially simplified, since the $\min(|x|, |y|)$ part always returns the scaled version of the variable that is not equal to $+\mathbb{U}_{max}$, resulting in the removal of a compare operation. The result of adding $+\mathbb{U}_{max}$ to a variable is not predefined and therefore this operation cannot be dropped. It is therefore preferable to find a condition where the input LLRs is 0 as this may simplify the equations used during decoding more efficiently as opposed to an input LLRs with a value of $+\mathbb{U}_{max}$.

Although only radix-2 BP decoding was used in [9], radix-4 BP decoding follows the same principle, as this property of BP decoding is independent of the radix used. Nonetheless, these simplifications will effect the radix-4 BP algorithm more than the radix-2 BP algorithm, since the equations for the radix-4 algorithm are greater in number as in complexity. This can potentially result in a bigger simplification of the radix-4 BP algorithm, with the disadvantage that there are many more combinations with specific input LLRs that can be skipped, and thus has a huge optimization space.

To explore the optimization space, a Matlab implementation was made of a radix-4 BP decoder. The frequency of occurrence was analyzed for any combination of input values 0, $+\mathbb{U}_{max}$ or any other value for any CU in the signal to noise

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on March 13,2024 at 14:44:51 UTC from IEEE Xplore. Restrictions apply.

TABLE I. Frequency of input values of $+\mathbb{U}_{max}$ or 0 for all CUs in a $\mathcal{P}(1024, 512)$ radix-4 BP polar code decoder using 10000 random code words and 15 iterations.

	Left-bound				**Right-bound**			
	0		$+\mathbb{U}_{\mathbf{max}}$		**0**		$+\mathbb{U}_{\mathbf{max}}$	
SNR (dB)	0.0	4.0	0.0	4.0	0.0	4.0	0.0	4.0
$\mathbf{L_e}$	5.3%	0.1%	0.8%	43.8%	12.9%	6.8%	0.8%	46.2%
$\mathbf{L_f}$	5.3%	0.1%	0.7%	44.7%	12.9%	6.8%	0.7%	47.3%
$\mathbf{L_g}$	5.2%	0.1%	0.8%	44.8%	12.8%	6.8%	0.8%	47.3%
$\mathbf{L_h}$	5.3%	0.1%	0.7%	44.9%	12.8%	6.8%	0.8%	47.5%
$\mathbf{R_a}$	25.5%	20.1%	16.8%	51.9%	29.5%	23.4%	20.1%	50.1%
$\mathbf{R_b}$	39.3%	33.1%	14.0%	46.1%	44.4%	39.6%	17.5%	42.4%
$\mathbf{R_c}$	35.8%	27.7%	14.7%	47.3%	40.9%	32.9%	18.2%	43.6%
$\mathbf{R_d}$	51.5%	39.7%	12.1%	22.1%	55.5%	47.6%	15.2%	27.6%

TABLE II. Frequency of value 0 combinations for R inputs of all CUs for a $\mathcal{P}(1024, 512)$ radix-4 BP polar code decoder using 10000 random code words and 15 iterations.

	Right-bound			
	0		$+\mathbb{U}_{\mathbf{max}}$	
SNR (dB)	0.0	4.0	0.0	4.0
$\mathbf{R_a}$	29.5%	23.4%	25.5%	20.1%
$\mathbf{R_a \cap R_b}$	28.9%	23.4%	24.6%	20.0%
$\mathbf{R_a \cap R_c}$	28.4%	23.3%	24.1%	20.0%
$\mathbf{R_a \cap R_d}$	29.3%	23.4%	25.0%	20.1%
$\mathbf{R_a \cap R_b \cap R_c}$	28.3%	47.6%	51.5%	39.7%
$\mathbf{R_b \cap R_c \cap R_d}$	38.3%	32.7%	32.1%	27.5%
$\mathbf{R_a \cap R_b \cap R_c \cap R_d}$	28.3%	23.3%	24.0%	20.0%

ratio (SNR) range between 0.0 and 4.0 dB. For the sake of clarity, Table I only shows the result of the SNR 0.0 and 4.0. The results of the other SNRs are in between these reported extremes.

A condition for an efficient reduction, which also simplifies the decoding equations as much as possible, must occur often. When emphasis is placed on input values equal to 0, the results in Table I indicate that this is less common for L inputs than for R inputs. Combinations of multiple R entries equal to 0 simplify the decoding equations more than just a single R that is 0. An analysis of these combinations is presented in Table II. The $R_b \cap R_c \cap R_d = 0$ condition is on average the most common of all SNRs tested, resulting in the following equations:

$$L_a = h(L_e, h(L_h, f(L_g, L_f))), \tag{19}$$
$$L_b = h(L_h, L_f) + h(L_e, h(L_g, R_a)), \tag{20}$$
$$L_c = h(L_h, L_g) + h(L_e, h(L_f, R_a)), \tag{21}$$
$$L_d = L_h + h(L_e, h(R_a, h(L_g, L_f))), \tag{22}$$
$$R_e = h(R_a, h(L_h, h(L_g, L_f))), \tag{23}$$
$$R_f = h(R_a, h(L_e, h(L_h, L_g))), \tag{24}$$
$$R_g = h(R_a, h(L_e, h(L_h, L_f))), \tag{25}$$
$$R_h = h(R_a, h(L_e, h(L_g, L_f))). \tag{26}$$

The results also show that the condition $R_a \cap R_b \cap R_c \cap R_d = 0$ is a preferable choice over $R_a \cap R_b \cap R_c = 0$, as the first highly simplifies the decoding equations to the following:

$$L_a = f(L_e, f(L_h, f(L_g, L_f))), \tag{27}$$
$$L_b = f(L_h, L_f), \tag{28}$$
$$L_c = f(L_h, L_g), \tag{29}$$
$$L_d = L_h, \tag{30}$$
$$R_e = R_f = R_g = R_h = 0. \tag{31}$$

Although the chosen condition $R_a \cap R_b \cap R_c \cap R_d = 0$ occurs less often (between 7.5% and 10% less), this condition is preferred due to the highly simplified decoding equations.

A similar condition is used for the radix-2 BP decoding equations. Here we use the condition $R_a \cap R_b = 0$, which results in the following simplified decoding equations:

$$L_a = f(L_c, L_d), \tag{32}$$
$$L_b = L_d, \tag{33}$$
$$R_c = R_d = 0. \tag{34}$$

The chosen conditions and the simplified equations are then incorporated into radix-2 and radix-4 BP decoders, where the algorithms are similar to the baseline algorithms Alg. 1 and Alg. 2 respectively.

VI. Performance analysis and comparison

A. Methodology

An implementation of the proposed algorithm is validated and compared against a textbook BP [11] baseline algorithm. For the proposed algorithm, the simplified equations, as given in Sec. V, are implemented. If the condition is true, the accompanying simplified equation is executed and no further actions are taken for this node.

To evaluate the performance of the baseline and proposed decoders, a Monte Carlo simulation is performed with 10000 random vectors u which are encoded into $\mathcal{P}(1024, 512)$ polar code codewords using a systematic polar code encoder. Before transmission, the codewords are modulated with binary phase-shift keying (BPSK) and transmitted over an additive white Gaussian noise (AWGN) channel at different noise levels (SNR).

For this evaluation, we are using Texas Instruments Hercules™ RM46x LaunchPad Development Kit, fitted with an ARM Cortex-R4 processor. It incorporates performance counters, e.g. cycle count and branch misses, which have been used to gather the results. As the compiler, we have used Texas Instruments *Code Composer Studio Version 12.1.0.00007* with compiler version *TI v20.2.7.LTS*. The compiler settings are:

```
-mv7R4 --code_state=32 -me -O4
--float_support=VFPv3D16 -g
--opt_for_speed=5 --fp_mode=strict
--symdebug:dwarf_version=4 --c99
--printf_support=nofloat
--diag_warning=225 --diag_wrap=off
--display_error_number
```

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on March 13,2024 at 14:44:51 UTC from IEEE Xplore. Restrictions apply.

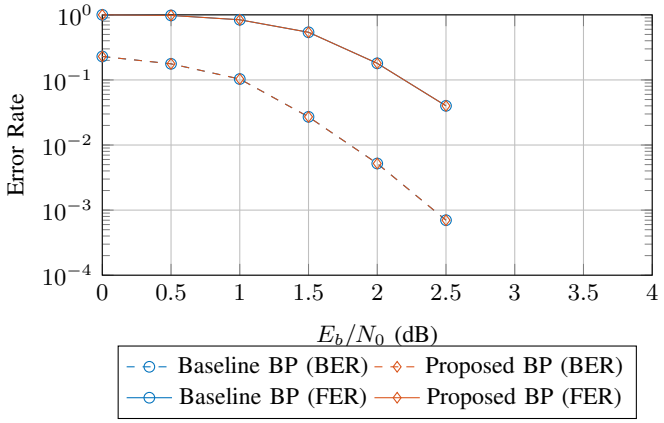

Fig. 4. BER and FER results for the radix-2 BP and proposed algorithms. A $\mathcal{P}(1024, 512)$ polar code is used.

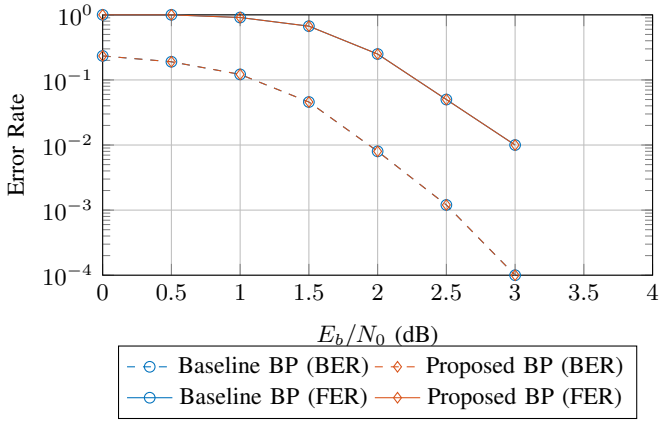

Fig. 5. BER and FER results for the radix-4 BP and proposed algorithms. A $\mathcal{P}(1024, 512)$ polar code is used.

```
--enum_type=packed --abi=eabi
```

The linker settings are the same as the compiler settings, but with the addition of:

```
-z -m"BP.map" --heap_size=0xA000
--stack_size=0x800 --reread_libs
--diag_wrap=off --display_error_number
--warn_sections --rom_model
--xml_link_info="BP_linkInfo.xml"
```

Finally, the received signal is demodulated and the LLRs are computed. The LLRs are processed by the baseline and proposed BP decoders using 15 iterations. After decoding the received signal, the estimated vector $\hat{u}$ is compared with the sent vector u, which finally results in frame error rate (FER) and bit error rate (BER) plots. At the same time several other statistics are gathered, like average cycle count, average number of branch misses and average code execution time.

B. Proposed algorithm compared to the baseline algorithm

In terms of BER and FER, we see in Fig. 4 and Fig. 5 that the baseline and proposed radix-2 and radix-4 BP decoders have equal performance.

When looking at the additional information gathered during the decoding of the given SNR range, we see that the proposed algorithms with the simplified equations are performing better than the baseline algorithms. In Tab. III, a breakdown of the cycle count, the number of branch misses, and the computed execution time is given, showing that the proposed algorithms perform approximately 15.4% and 22.04% faster for the radix-2 and radix-4 decoders respectively. The increased decoding throughput comes at the cost of approximately 2893% and 1277% more branch misses for the radix-2 and radix-4 decoders respectively, because branches are introduced in order to choose between the simplified and the full equations.

TABLE III. Breakdown of the average cycle count, number of branch misses (B.M.) and execution time (E.T.) for the radix-2 and radix-4 baseline BP and the proposed algorithms.

	Radix-2			Radix-4		
Algorithm	**baseline**	**proposed**	**Δ**	**baseline**	**proposed**	**Δ**
Iterations	15		0%	15		0%
Cycles (≈)	6317918	5345910	−15.4%	10024194	7814510	−22.04%
B.M. (≈)	310	9281	2893%	158	2175	1277%
E.T. (≈ms)	114.87	97.20	−15.4%	182.26	142.08	−22.04%

VII. Conclusion and Future Work

The preliminary results in [9], using the instruction set and cycle timings of an ARM Cortex-R4 processor, had shown that a throughput gain of $\approx 13\%$ could be expected for a radix-2 BP decoder using the simplified equations. In this paper we have implemented and compared radix-2 and radix-4 simplifications for the equations used in the belief propagation (BP) algorithm. The proposed simplified algorithms increase throughput by 15.4% and 22.04% for a radix-2 and radix-4 BP decoder respectively compared to the baseline BP algorithm, with equal bit and frame error rate.

As future work we would like to explore the net benefits of the reduced number of operations on energy consumption on the used processor and other processors.

Acknowledgement

This work is part of the research program Perspectief ZERO with project number P15-06 Project 4, which is (partly) financed by the Dutch Research Council (NWO).

References

[1] C. Berrou, A. Glavieux, P. Thitimajshima, "Near Shannon limit error-correcting coding and encoding: Turbo-codes (1)," *IEEE Int. Conf. Commun.*, no. 1, pp. 1064–1070, 1993.

[2] R. Gallager, "Low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 8, no. 1 , pp. 21–28, jan 1962.

[3] E. Arikan, "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, 2009.

[4] E. Şaşoğlu, E. Telatar, E. Arikan, "Polarization for arbitrary discrete memoryless channels," *2009 IEEE Inf. Theory Work. ITW 2009*, vol. 2, pp. 144–148, 2009.

[5] V. Bioglio, C. Condo, I. Land, "Design of polar codes in 5G new radio," *IEEE Commun. Surv. Tutorials*, vol. 23, no. 1, pp. 29–40, 2021.

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on March 13,2024 at 14:44:51 UTC from IEEE Xplore. Restrictions apply.

[6] A. Alamdar-Yazdi, F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Commun. Lett.*, vol. 15, no. 12, pp. 1378–1380, 2011.
[7] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, W. J. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 946–957, 2014.
[8] O. Meteer, A. van den Brink, M. J. G. Bekooij, "Energy-efficient radix-4 belief propagation polar code decoding using an efficient sign-magnitude adder and clock gating," in *2022 25th Euromicro Conf. Digit. Syst. Des.* IEEE, aug 2022, pp. 126–133.
[9] A. B. van den Brink, M. J. G. Bekooij, "Computational complexity reduced belief propagation algorithm for polar code decoders," in *2021 Asia-Pacific Signal Inf. Process. Assoc. Annu. Summit Conf. (APSIPA ASC)* Tokyo, Japan: IEEE, dec 2021, pp. 318-323.
[10] A. B. van den Brink, M. J. G. Bekooij, "Enhanced loop-weakened belief propagation algorithm for performance enhanced polar code decoders," in *2021 Asia-Pacific Signal Inf. Process. Assoc. Annu. Summit Conf. (APSIPA ASC)* Tokyo, Japan: IEEE, dec 2021, pp. 299-304.
[11] E. Arikan, "Polar codes : A pipelined implementation," *4th Int. Symp. Broadband Commun. (ISBC 2010)*, pp. 11–14, 2010.
[12] A. Eslami, H. Pishro-Nik, "On bit error rate performance of polar codes in finite regime," *2010 48th Annu. Allert. Conf. Commun. Control. Comput. Allert. 2010*, pp. 188–194, 2010.
[13] J. Sha, J. Liu, J. Lin, Z. Wang, "A stage-combined belief propagation decoder for polar codes," *J. Signal Process. Syst.*, vol. 90, no. 5, pp. 687–694, 2018.
[14] E. Arikan, "Systematic polar coding," *IEEE Commun. Lett.*, vol. 15, no. 8, pp. 860–862, 2011.
[15] I. Tal, A. Vardy, "List decoding of polar codes," *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.
[16] K. Niu, K. Chen, J. Lin, Q. T. Zhang, "Polar codes: Primary concepts and practical decoding algorithms," *IEEE Commun. Mag.*, vol. 52, no. July, pp. 192–203, 2014.
[17] A. Elkelesh, M. Ebada, S. Cammerer, S. T. Brink, "Belief propagation list decoding of polar codes," *IEEE Commun. Lett.*, vol. 22, no. 8, pp. 1536–1539, 2018.
[18] C. Albayrak, C. Simsek, K. Turk, "Low-complexity early termination method for rateless soft decoder," *IEEE Commun. Lett.*, vol. 21, no. 11, pp. 2356–2359, 2017.
[19] Y. Ren, C. Zhang, X. Liu, X. You, "Efficient early termination schemes for belief-propagation decoding of polar codes," *Proc. - 2015 IEEE 11th Int. Conf. ASIC, ASICON 2015*, 2016.
[20] C. Simsek, K. Turk, "Simplified early stopping criterion for belief-propagation polar code decoders," *IEEE Commun. Lett.*, vol. 20, no. 8, pp. 1515–1518, 2016.
[21] S. Sun, Z. Zhang, "Architecture and optimization of high-throughput belief propagation decoding of polar codes," in *Proc. - IEEE Int. Symp. Circuits Syst.*, vol. 2016-July, pp. 165–168, 2016.
[22] Y. S. Park, Y. Tao, S. Sun, Z. Zhang, "A 4.68Gb/s belief propagation polar decoder with bit-splitting register file," *IEEE Symp. VLSI Circuits, Dig. Tech. Pap.*, pp. 2013–2014, 2014.
[23] A. B. van den Brink, M. J. G. Bekooij, "Conflict-free vectorized in-order in-place radix-r belief propagation polar code decoder algorithm," *ACM Int. Conf. Proceeding Ser.* New York, NY, USA, ACM, apr 2020, pp. 18–23.

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on March 13,2024 at 14:44:51 UTC from IEEE Xplore. Restrictions apply.