Conflict-Free Vectorized In-order In-place Radix-r Belief Propagation Polar Code Decoder Algorithm

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

Marco J.G. Bekooij*†
† NXP Semiconductors
Department of Embedded Software
and Signal Processing
Eindhoven, The Netherlands
marco.bekooij@nxp.com

ABSTRACT

A vectorized belief propagation polar code decoder is desirable because of the potentially high throughput and the ability of integration in processors that perform vectorized processing and access wide memory words. However, current state-of-the-art belief propagation polar code decoder algorithms do not perform vector processing and store intermediate results in non consecutive memory locations. Also the current state-of-the-art belief propagation polar code decoders require separate memories to store left and right bound intermediate results.

In this paper we propose a vectorized in-order in-place belief propagation polar code decoder algorithm where all stages access vectorized data from memory. This results in a high throughput because vectors of elements can be fetched from and stored in memory in each clock cycle. Our algorithm also accommodates for per stage in-place computations which halves the required internal memory. Furthermore, the algorithm has a regular memory addresses access pattern. Conflict free vectorized memory access is achieved by making use of transpose operations on small groups of intermediate results. The use of the transpose operations also results in that both input and output results are placed on subsequent locations in memory.

CCS Concepts

• **Theory of computation → Vector / streaming algorithms**; **Shared memory algorithms**.

Keywords

Conflict-free; Vectorized; Belief Propagation; Polar Code

ICCBN '20, April 15–18, 2020, Auckland, New Zealand

ACM ISBN 978-1-4503-7504-7/20/04…$15.00
https://doi.org/10.1145/3390525.3390539

1 INTRODUCTION

Since the introduction, polar codes [3] have attracted much attention for their provable capacity-achieving error correction capability for discrete memoryless channels with low decoding complexity as the block length reaches infinity. Originally, successive cancellation (SC) was proposed for decoding polar codes with a computational complexity $O(N \log N)$, with N being the length of the code. As computational complexity and decoding latency becomes a bottleneck when the block length increases, the use of shorter block length is desired, especially for resource constrained applications such as Internet of things (IoT) devices, or latency critical applications, e.g. vehicle-to-X communication (V2X).

In most of these applications short messages are used, hence short block lengths. However, the decoding performance of SC degrades with decreasing block lengths since the polarization effect cannot be fully utilized. To improve the decreasing performance, several other decoding algorithms were proposed at the expense of an increase in computational complexity and decoding latency. As proposed in [19] successive cancellation list (SCL) decoding improves the performance of the SC decoder at the cost of a higher computational complexity. However, the performance offset is still worse compared to the state-of-the-art low-density parity check (LDPC) and Turbo codes. Instead of selecting the most probable codeword from the list, the performance of the SCL decoder can outperform LDPC and Turbo code decoders by using systematic encoding [4] and a cyclic redundancy check (CRC) checksum [11] for selecting the correct codeword from the list [19]. Despite that an enhanced SCL decoder can outperform LDPC and Turbo decoders, a bottleneck arises for high-speed real-time applications due to the serial nature of the SCL algorithm.

The belief propagation (BP) decoding algorithm[5] can potentially address the high decoding latency of the SCL algorithm. Since the BP decoding algorithm is an iterative message passing algorithm, it introduces scattering of the messages (log likelihood ratios (LLRs)) between stages in memory as shown in Fig. 1.

In this paper we overcome the data scattering in memory of the scaled min-sum (SMS) BP decoder by introducing a generalized conflict-free vectorized in-order in-place algorithm for a radix-r BP decoder. Compared to the state-of-the-art SC and BP decoders, the proposed BP decoder algorithm accesses vectorized data from memory. Our algorithm also accommodates for per stage in-place computations which halves the internal memory required for the intermediate results per stage. The algorithm achieves conflict free

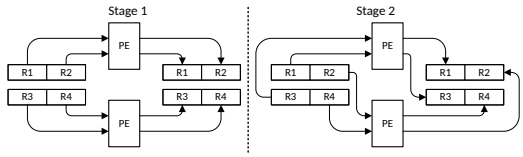

Figure 1: Example of scattered data in memory. In Stage 1 all vector elements are stored in the position they are read from. In Stage 2 the data is then scattered across vectors in a undesirable manner.

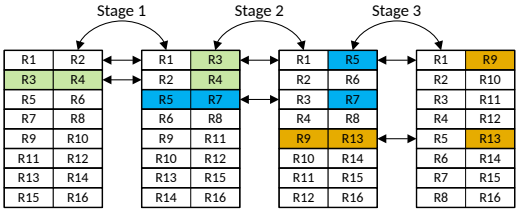

Figure 2: $N = 16$ memory locations of LLR messages between stages

vectorized memory access by making use of the transpose operation on small groups of vectors of intermediate results. The use of the transpose operations also results in that both input and output results are placed on subsequent locations in memory.

As state-of-the-art polar decoders always have in-order inputs and bit-reversed outputs (or vice versa), the proposed algorithm keeps the data at both the input and the output of the decoder in-order on subsequent locations in memory as shown in Fig. 2.

Compared to state-of-the-art SC and BP decoders, the proposed vectorized in-place algorithm reduces the amount memory accesses required for the decoding, hence the decoder can achieve a higher throughput when implemented on a DSP with SIMD instruction support.

The remainder of this paper is organized as follows. Section 2 gives an overview of related work. Section 3 describes the preliminaries of polar codes and gives a description of belief propagation decoding. The proposed radix-r conflict-free vectorized memory algorithm is addressed in Section 5. Section 6 provides the simulation results. Finally, section 7 concludes this paper.

2 RELATED WORK

Polar codes are introduced in [3]. In this paper it is proven that polar codes achieve the capacity of any binary-input discrete memoryless channel (B-DMC). Due to this and other properties, a lot of research has been done in the last decade, for example in [4, 11, 19]. Although capacity-achieving was proved for infinite block length SC decoding, the error performance for finite block length is not guaranteed.

A inherently parallelizable algorithm for polar codes is the BP decoding algorithm first described in [2]. Recent work [20] proposed an efficient BP decoder by reducing the critical path delay. The authors in [8] proposed an improvement in performance by introducing a BP list decoder. In [1, 13, 16] the authors proposed different early stopping criteria to reduce the number of iterations needed in the BP decoding algorithm, therefore reducing the decoding latency. In [18] the authors proposed architectural optimizations to increase the throughput of the BP decoder. A hardware BP polar code decoder is introduced in [12] which reduces the required memory to store the intermediate LLRs. In [15] the number of read and write operations are reduced by proposing to combine stages of the BP decoder. Our work differs from the aforementioned studies because it addresses the memory bottleneck of the belief propagation algorithm by allowing intermediate results to be read and written as vectors in wide memory words.

As for implementing Fast Fourier Transforms (FFTs) on Single Instrustion Multiple Data (SIMD) instruction digital signal processing (DSP) processors [6, 14, 17, 21], to the best of our knowledge, no such work is performed on BP polar code decoders. Therefore, our work provides not only a novel hardware solution, but also a novel approach for the implementation of BP polar code decoders on DSPs, which allow the definition of application specific SIMD instructions for Polar code decoding, like in the Cadence Tenscilica ConnX family DSPs.

3 PRELIMINARY

3.1 BP Polar Decoding

Polar code is a method for constructing a capacity achieving linear block code on a symmetric binary-input discrete memoryless channel such as the binary symmetric channel (BSC). Like the definition in [3] we assume a polar code identified by a parameter vector $(N, K, \mathcal{A}, u_{\mathcal{A}^c})$, where $N = 2^n$ is the length of the codeword, K the number of information bits, $\mathcal{A}$ the set of information bits, and $u_{\mathcal{A}^c}$ represent the frozen bits. The polar encoding and decoding process consists of the following three steps.

(1) The source vector $u_1^N = (u_1, u_2, \cdots, u_N)$ is encoded as follows:
$$x_1^N = u_1^N G_N = u_1^N B_N F^{\otimes n} \tag{1}$$
where G_N is the generator matrix, B_N is a permutation matrix known as a bit-reversal matrix, and $F^{\otimes n}$ is the *Kronecker* power of F defined as follows:
$$F^{\otimes n} = F \otimes F^{\otimes (n-1)} \tag{2}$$
where $n = \log_2 N$, and $F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$

(2) x_1^N is sent over the channels W^N, where the channel output $y_1^N = (y_1, y_2, \cdots, y_N)$ is presented to the decoder.

(3) Using the knowledge of the polar code $\mathcal{A}$ and $u_{\mathcal{A}^c}$ and the given channel output y_1^N the decoder estimates $\hat{u}_1^N$ of u_1^N.

Based on the factor graph representation, a BP decoder for polar codes [10] is constructed. Fig. 3 shows an example of the factor graph for the case of an $N = 8$ BP polar decoder. Each stage consists of $N/2$ computational units (CUs). The CU has four terminals as shown in Fig. 4. The terminals maps to:

$$\begin{aligned} L_{i,j}^{(m+1)} &= f(L_{i+1,2j-1}^{(m)}, L_{i+1,2j}^{(m)} + R_{i,j+N/2}^{(m)}) \\ L_{i,j+N/2}^{(m+1)} &= f(R_{i,j}^{(m)}, L_{i+1,2j-1}^{(m)}) + L_{i+1,2j}^{(m)} \\ R_{i+1,2j-1}^{(m+1)} &= f(R_{i,j}^{(m)}, L_{i+1,2j}^{(m)} + R_{i,j+N/2}^{(m)}) \\ R_{i+1,2j}^{(m+1)} &= f(R_{i,j}^{(m)}, L_{i+1,2j-1}^{(m)}) + R_{i,j+N/2}^{(m)} \end{aligned} \tag{3}$$

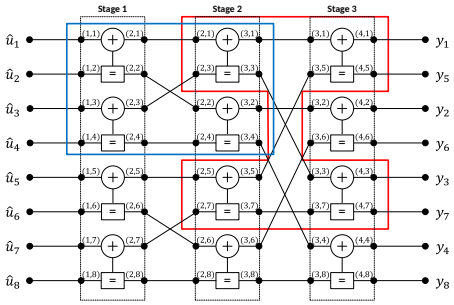

Figure 3: Polar code factor graph with $N = 8$.

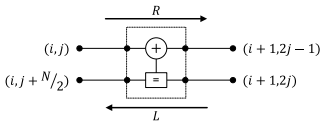

Figure 4: BP computational unit (CU) factor graph.

where m is the iteration and the f function is calculated according to:

$$\begin{aligned} f(a,b) &= 2\tanh^{-1}(\tanh(a/2)\tanh(b/2)) \\ &\approx 0.9 \cdot \text{sign}(a) \cdot \text{sign}(b) \cdot \min(|a|,|b|) \end{aligned} \tag{4}$$

where the approximation is the min-sum approximation similar to [9]. The four terminals are labeled with integers (i, j), $1 \le i \le n-1$, $1 \le j \le N/2$, where i and j indicates the stage number and node number respectively. The BP decoding is an iterative, message passing process through the factor graph. There are two types of messages involved: left bound messages L and right bound messages R. Each message of which the data, represented by LLRs, is initially assigned an LLR depending on the side at which the node is located. The left most R messages are assigned a value using (5) and the right most L messages are assigned a value with the channel LLR using (6). All other nodes are initially set to zero.

$$R_{1,j} = \begin{cases} 0 & \text{if } j \in \mathcal{A} \\ \infty & \text{if } j \in \mathcal{A}^c \end{cases} \tag{5}$$

$$L_{n+1,j} = \ln \frac{\Pr(y_j \mid x_j = 0)}{\Pr(y_j \mid x_j = 1)} \tag{6}$$

After m iterations, $\hat{u}_1^N$ can be decided at the left most terminals using (7).

$$\hat{u}_j = \begin{cases} 0 & \text{if } L_{1,j} + R_{1,j} \ge 0 \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

4 BASIC IDEA

To reduce the number of memory accesses, instead of reading or writing every single data element from or to memory, all the required data elements could be vectorized.

Lets assume a part of the factor graph of an $N = 8$ BP polar decoder (Fig. 3 (blue box)) using a radix-2 BP CUs, as shown in Fig. 1. For the sake of clarity, we only show the right bound messages computations. After vectorizing the initial message $R1$ to $R4$ into 2

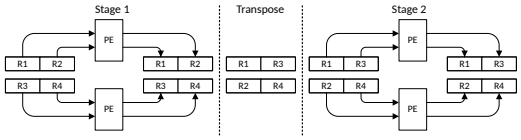

Figure 5: Example of non-scattered data in memory

vectors $\bar{r}_1 = [R1, R2]$ and $\bar{r}_2 = [R3, R4]$, the upper CU, in the first stage, reads and processes vector $\bar{r}_1$. The computed results can then be vectorized and stored in memory. The lower CU reads and processes vector $\bar{r}_2$ and stores the vectorized results in memory. After all vectors in stage 1 are processed, the next stage is activated. In stage 2 the upper CU has to process the message $R1$ and $R3$, but these massages are positioned in two different vectors, therefore this CU needs to read 2 vectors (4 elements) and will discard half of the read elements, because $R2$ and $R4$ are not required for the computation. The same holds for the lower CU in stage 2.

In order to overcome this scattering of the elements in memory, a transpose operation of two consecutive vectors between the two stages, as shown in Fig. 5 is performed. Therefore, vectors read in the next stage contain only the needed elements and hence no unnecessary vectors have to be read and no data is discarded.

5 PROPOSED CONFLICT-FREE, IN-PLACE, RADIX-r, VECTORIZED MEMORY ALGORITHM

5.1 Conflict-free, Radix-r, Vectorized Algorithm

Lets first assume a polar code as proposed by Arikan [3]. This radix-2 algorithm assumes $N = 2^n$ for any $n \ge 0$ in the following. Let I_m be the m-dimensional identity matrix for any $m \ge 1$. The recursive definition of G_N in (1) in algebraic form is given by:

$$G_N = (I_{N/2} \otimes F) R_N (I_2 \otimes G_{N/2}), \text{ for } N \ge 2 \tag{8}$$

with $G_1 = I_1$ and R_N is the "reverse shuffle" or bit-reversal operator. By algebraic manipulation, a second recursive formula is obtained:

$$\begin{aligned} G_N &= R_N (F \otimes I_{N/2})(I_2 \otimes G_{N/2}) \\ &= R_N (F \otimes G_{N/2}) \end{aligned} \tag{9}$$

for $N \ge 2$. By substituting $G_{N/2} = R_{N/2}(F \otimes G_{N/4})$ into (9) we get:

$$\begin{aligned} G_N &= R_N (F \otimes (R_{N/2}(F \otimes G_{N/4}))) \\ &= R_N (I_2 \otimes R_{N/2})(F^{\otimes 2} \otimes G_{N/4}) \end{aligned} \tag{10}$$

Repeating the recursive substitution, we obtain:

$$G_N = B_N F^{\otimes n} \tag{11}$$

where $B_N \triangleq R_N (I_2 \otimes R_{N/2})(I_4 \otimes R_{N/4}) \cdots (I_{N/2} \otimes R_2)$. From the definition of B_N it can be seen that:

$$B_N = R_N (I_2 \otimes B_{N/2}) \tag{12}$$

According to (11) and (12) and the recursive nature of R_N (bit-reversal operator), at every next stage in the factor graph the $N/2$ inputs are in bit reverse order with respect to the previous stage. The radix-2 bit-reversal between stages is shown in the factor graph (blue box) in Fig. 3. In the first stage the upper and lower CUs will

compute the LLRs $(l_{2,1}, l_{2,2})$ and $(l_{2,3}, l_{2,4})$ respectively. In the next stage the upper and lower CUs need to consume LLRs $(l_{2,1}, l_{2,3})$ and $(l_{2,2}, l_{2,4})$ respectively. Lets assume that every pair of LLRs is a row in a matrix, we get:

$$S1_{out} = \begin{bmatrix} l_{2,1} & l_{2,2} \\ l_{2,3} & l_{2,4} \end{bmatrix}, \quad S2_{in} = \begin{bmatrix} l_{2,1} & l_{2,3} \\ l_{2,2} & l_{2,4} \end{bmatrix} \tag{13}$$

where $S1_{out}$ is the output matrix for stage 1 and $S2_{in}$ the input matrix for stage 2. From (13) it is trivial that:

$$S2_{in} = S1_{out}^{T} \tag{14}$$

This connection between bit-reversal and the matrix transpose is shown in [7].

Now lets assume a radix-r polar code like above for $N = r^n$ for any $n \geq 0$ and $r \geq 2$. The recursive definition of G_N in (8) is now given by:

$$G_N = (I_{N/r} \otimes F) R_N (I_r \otimes G_{N/r}), \text{ for } N \geq r \tag{15}$$

After applying all the steps as above the general radix-r formulation is as follows:

$$G_N = B_N F^{\otimes n} \tag{16}$$

where $B_N \triangleq R_N(I_r \otimes R_{N/r})(I_r^2 \otimes R_{N/r^2}) \cdots (I_{N/r} \otimes R_r)$. From the definition of B_N it can be seen that:

$$B_N = R_N(I_r \otimes B_{N/r}). \tag{17}$$

The radix-r bit-reversal between stages is maintained, hence:

$$S1_{out} = \begin{bmatrix} l_{2,1} & \cdots & l_{2,r} \\ \vdots & \ddots & \vdots \\ l_{2,r^2-r+1} & \cdots & l_{2,r^2} \end{bmatrix} \tag{18}$$

$$S2_{in} = \begin{bmatrix} l_{2,1} & \cdots & l_{2,r^2-r+1} \\ \vdots & \ddots & \vdots \\ l_{2,r} & \cdots & l_{2,r^2} \end{bmatrix} \tag{19}$$

From Eqs. 18 and 19 it can be concluded, that:

$$S2_{in} = S1_{out}^{T} \tag{20}$$

Based on the observation in (20) and the example given in Fig. 6, a vectorized, conflict free, memory efficient BP decoding algorithm for polar codes is proposed in Alg. 1. The final stage (stage 3 in the example) will not perform a transpose afterwards, because there is no next stage requiring the transposition of the data. It should also be noted that in every stage in the example, the vectors are not necessarily on subsequent addresses. In every stage, starting from stage 1 and except the final stage, the vector addresses are $2^{stage-1}$ apart.

5.2 In-place Algorithm

A single iteration of the BP polar decoder is defined by first a right bound message propagation for $n-1$ stages and then a left bound message propagation for n stages totaling $2n-1$ stages. In each stage a vector is read from both R and L memory and a single vector is written into either the R or L memory. Due to this unidirectionality (only R or L messages are computed and stored, Tab. 1) both R and L messages are allowed to share the same memory space to store the intermediate messages (Tab. 2), reducing the memory for storing the intermediate message by 50%.

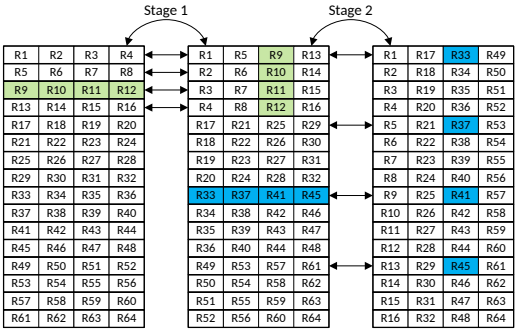

Figure 6: Radix-4 in-order vectorized right message memory contents ($N = 64$)

Table 1: Read and write operations for separate intermediate memory

	Right Bound				Left Bound			
Stage	1	2	⋯	n-1	n	n-1	⋯	1
Read R	R_0	R_1	⋯	R_{n-1}	R_{n-1}	R_{n-2}	⋯	R_0
Read L	L_1	L_2	⋯	L_n	L_n	L_{n-1}	⋯	L_1
Write R	R_1	R_2	⋯	R_n	-	-	⋯	-
Write L	-	-	⋯	-	L_{n-1}	L_{n-2}	⋯	L_0

Table 2: Read and write operations for shared intermediate memory

	Right Bound				Left Bound			
Stage	1	2	⋯	n-1	n	n-1	⋯	1
Read	L_1	L_2	⋯	L_n	R_{n-1}	R_{n-2}	⋯	R_0
Write	R_1	R_2	⋯	R_n	L_{n-1}	L_{n-2}	⋯	L_0

5.3 In-order Algorithm

Both Fig. 3 and Fig. 6 clearly show that the elements are in-order row vectors at the leftmost matrix and in-order column vectors at the rightmost matrix. The in-order property for the input and output of the algorithm holds for any code block length and for any decoder radix.

5.4 Address Calculation Algorithm

In [17] a method for generating the addressing sequence required by FFT algorithms is proposed. Because Polar Codes exhibit a similar butterfly structure, we use the same approach in our algorithm.

Lets assume consecutive memory addresses for all vectors in the intermediate memory block as shown in Fig. 7. In the first stage of this radix-2 $N = 8$ decoder, the distance between two consecutive vectors required is 1. For every next stage the difference between the two required vectors is 2 times the difference of the previous stage, i.e. 1, 2 and 4 respectively.

For encoding 32 address a total of 5 bits are required. For each stage 8 vectors have to be read as shown in Tab. 3. By looking at the sequences the following patterns can be noted. In each stage the first two bits of the addresses are “00”, “01” and “10” respectively, hence encoding the stages as $s = [0, 1, \cdots, n-1]$. The remaining

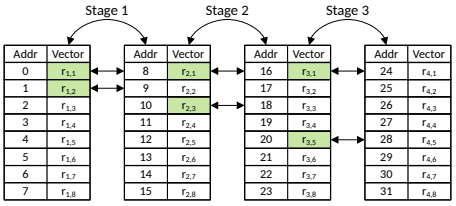

Figure 7: Polar code $N = 8$ radix-2 address sequence

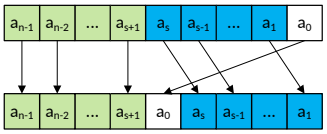

Figure 8: Partial bit rotation

Table 3: Address sequence, $N = 16$ radix-2

Stage Address Offset	Stage		
	1	2	3
0	0 $00000_{(2)}$	8 $01000_{(2)}$	16 $10000_{(2)}$
1	1 $00001_{(2)}$	10 $01010_{(2)}$	20 $10100_{(2)}$
2	2 $00010_{(2)}$	9 $01001_{(2)}$	17 $10001_{(2)}$
3	3 $00011_{(2)}$	11 $01011_{(2)}$	21 $10101_{(2)}$
4	4 $00100_{(2)}$	12 $01100_{(2)}$	18 $10010_{(2)}$
5	5 $00101_{(2)}$	14 $01110_{(2)}$	22 $10110_{(2)}$
6	6 $00110_{(2)}$	13 $01101_{(2)}$	19 $10011_{(2)}$
7	7 $00111_{(2)}$	15 $01111_{(2)}$	23 $10111_{(2)}$

bits are a permutation of the bits of the stage address offsets $a = [0, 1, \cdots, N-1]$.

The stage address offset permutation is shown in Fig. 8. Depending on which stage s is processed, up to bit s of the offset address a, the bits are rotated 1-bit to the right. The remaining bits are unchanged. Finally, the required vector address is composed by concatenating the bits of s and a (see Alg. 2).

6 PERFORMANCE ANALYSIS AND COMPARISON

To store the messages needed for a radix-r BP decoder a total number of $N \log_r N$ memory elements for the right bound messages and $N(\log_r N + 1)$ memory elements for the left bound messages are needed, hence the cumulative total of memory elements is $N(2 \log_r N + 1)$.

In a non-vectorized radix-2 BP decoder each CU will read 4 LLRs (2 right bound and 2 left bound LLRs) to compute 2 new LLRs needed by the next stage as shown in Fig. 4. Every stage in the factor graph is composed out of $\frac{N}{2}$ CUs as shown in Fig. 3, where the right bound message and the left bound messages are computed over $\log_2(N) - 1$ and $\log_2(N)$ stages respectively, resulting in $6N \log_2(N) - 3N$ memory operations (read/writes) per iteration.

Table 4: A comparison of the number of memory operations and elements between a non-vectorized algorithm and our proposed algorithm for a 1024-bit Polar code

	BP[10]	Proposed	
		$r = 2$	$r = 4$
Operations	58368	29184	6912
Elements	21504	10752	2816

In the case of a vectorized radix-2 BP decoder, a vector of 2 LLRs can be read or written in a single memory operation. While the factor graph is not changed, the number of memory operations and elements are divided by 2, hence for the radix-2 vectorized decoder, the total number of memory operations per iteration is $3N \log_2(N) - \frac{3}{2}N$ resulting in a decrease of 50% in operations. Although the number of elements is halved, the elements are twice the size of the non-vectorized memory elements. In general the number of required memory operations for a radix-r vectorized implementation is $\frac{3N}{r}(2 \log_r N - 1)$ and the number of memory elements required is $\frac{N}{r}(2 \log_r N + 1)$. In Tab. 4 a comparison per iteration is made for a 1024-bit polar code using a non-vectorized BP decoder, and the proposed vectorized BP decoder.

The throughput of a BP decoder on a DSP is limited by the speed, data can be fetched from or stored in memory, in other words the memory bandwidth bound. By vectorizing with a factor of 2 the throughput will be doubled.

7 CONCLUSION

In this paper we have introduced a vectorized version of a belief propagation Polar code decoder. Vectorization improves the throughput in DSPs with SIMD instructions because wide memory words can be fetched. The proposed algorithm requires only $N/r(\log_r N + 1)$ memory elements compared to the $2N(\log_r N)$ elements in the non-vectorized BP implementation [10]. The memory access pattern is regular and can be computed with a rotate right of a linearly increasing index. This also holds for input and output samples which are stored in subsequent memory locations. This regular access pattern is achieved by making use of transpose operations on r^2 intermediate results of the computation units of a radix-r BP decoder.

Algorithm 1: Proposed radix-r, memory efficient BP decoder

1: Initialize $\mathbf{Mem}_{\mathbf{shared}}$ ▷ Shared message memory

2: **while** not $Iteration_{max}$ **do**
3: **for** $Stage \in 0, \cdots, n-2$ **do** ▷ Compute right bound
4: **for** $Vector \in 0, 2, \cdots, N/r - 1$ **do**
5: $Addr1 \leftarrow$ ComputeAddr(Stage, Vector)
6: $Addr2 \leftarrow$ ComputeAddr(Stage, Vector+1)
7: $R_{in} \leftarrow Mem_{Shared}(Addr1)$

```
            L_in ← Mem_Shared(Addr1 + N/r)
            M(1) ← ComputeLLR(R_in, L_in)

            R_in ← Mem_Shared(Addr2)
            L_in ← Mem_Shared(Addr2 + N/r)
            M(2) ← ComputeLLR(R_in, L_in)

            M ← M^T
            Mem_Shared(Addr1 + N/r) ← M(1)
            Mem_Shared(Addr2 + N/r) ← M(2)
        end for
    end for
    for Stage ∈ n − 1, ··· , 0 do                ▷ Compute left bound
        for Vector ∈ 0, 2, ··· , N/r − 1 do
            Addr1 ← ComputeAddr(Stage, Vector)
            Addr2 ← ComputeAddr(Stage, Vector+1)

            R_in ← Mem_Shared(Addr1)
            L_in ← Mem_Shared(Addr1 + N/r)
            M(1) ← ComputeLLR(R_in, L_in)

            R_in ← Mem_Shared(Addr2)
            L_in ← Mem_Shared(Addr2 + N/r)
            M(2) ← ComputeLLR(R_in, L_in)

            M ← M^T
            Mem_Shared(Addr1) ← M(1)
            Mem_Shared(Addr2) ← M(2)
        end for
    end for
    for Vector ∈ 0, 1, ··· , N/r − 1 do           ▷ Compute final
        Addr ← ComputeAddr(0, Vector)

        R_in ← Mem_Shared(Addr)
        L_in ← Mem_Shared(Addr + N/r)
        M ← ComputeLLR(R_in, L_in)

        Mem_Output(Vector) ← M
    end for
end while
```

Algorithm 2: Compute Memory Address

```
function ComputeAddr(Stage, Vector)
    S_bits ← Stage                               ▷ Convert to (n − 2) bits

    for j ∈ 0, 1, ··· , n − 1 do
        if j > Stage then
            Addr_bits(j) = Vector(j)
        else if j = Stage then
            Addr_bits(j) = Vector(0)
        else
            Addr_bits(j) = Vector(j + 1)
        end if
    end for
    return [S_bits Addr_bits]
end function
```

REFERENCES

[1] Cenk Albayrak, Cemaleddin Simsek, and Kadir Turk. 2017. Low-complexity early termination method for rateless soft decoder. *IEEE Commun. Lett.* 21, 11 (2017), 2356–2359. https://doi.org/10.1109/LCOMM.2017.2740207

[2] E. Arikan. 2008. A performance comparison of polar codes and Reed-Muller codes. *IEEE Commun. Lett.* 12, 6 (jun 2008), 447–449. https://doi.org/10.1109/LCOMM.2008.080017

[3] Erdal Arikan. 2009. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Trans. Inf. Theory* 55, 7 (2009), 3051–3073. https://doi.org/10.1109/TIT.2009.2021379 arXiv:0807.3917

[4] Erdal Arikan. 2011. Systematic polar coding. *IEEE Commun. Lett.* 15, 8 (2011), 860–862. https://doi.org/10.1109/LCOMM.2011.061611.110862 arXiv:arXiv:1011.1669v3

[5] Erdal Arıkan. 2010. Polar codes : A pipelined implementation. *4th Int. Symp. Broadband Commun. (ISBC 2010)* (2010), 11–14.

[6] Hoseok Chang and Wonyong Sung. 2008. Efficient vectorization of SIMD programs with non-aligned and irregular data access hardware. *Embed. Syst. Week 2008 - Proc. 2008 Int. Conf. Compil. Archit. Synth. Embed. Syst. CASES'08* (2008), 167–175. https://doi.org/10.1145/1450095.1450121

[7] Pierre Duhamel. 1990. A connection between bit reversal and matrix transposition: hardware and software consequences. *IEEE Trans. Acoust.* 38, 11 (1990), 1893–1896. https://doi.org/10.1109/29.103090

[8] Ahmed Elkelesh, Moustafa Ebada, Sebastian Cammerer, Stephan Ten Brink, and B E R Bler. 2018. Belief Propagation List Decoding of Polar Codes. *IEEE Commun. Lett.* 22, 8 (2018), 1536–1539. https://doi.org/10.1109/LCOMM.2018.2850772 arXiv:1806.10503

[9] Marc P.C. Fossorier, Miodrag Mihaljevic, and Hideki Imai. 1999. Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Trans. Commun.* 47, 5 (1999), 673–680. https://doi.org/10.1109/26.768759

[10] Frank R. Kschischang and Brendan J. Frey. 1998. Iterative decoding of compound codes by probability propagation in graphical models. *IEEE J. Sel. Areas Commun.* 16, 2 (1998), 219–230. https://doi.org/10.1109/49.661110

[11] Kai Niu and Kai Chen. 2012. CRC-aided decoding of polar codes. *IEEE Commun. Lett.* 16, 10 (2012), 1668–1671. https://doi.org/10.1109/LCOMM.2012.090312.121501

[12] Youn Sung Park, Yaoyu Tao, Shuanghong Sun, and Zhengya Zhang. 2014. A 4.68Gb/s belief propagation polar decoder with bit-splitting register file. In *IEEE Symp. VLSI Circuits, Dig. Tech. Pap.* 2013–2014. https://doi.org/10.1109/VLSIC.2014.6858413

[13] Yuanrui Ren, Chuan Zhang, Xing Liu, and Xiaohu You. 2016. Efficient early termination schemes for belief-propagation decoding of polar codes. *Proc. - 2015 IEEE 11th Int. Conf. ASIC, ASICON 2015* (2016). https://doi.org/10.1109/ASICON.2015.7517046

[14] Seyed A. Rooholamin and Sotirios G. Ziavras. 2015. Modular vector processor architecture targeting at data-level parallelism. *Microprocess. Microsyst.* 39, 4-5 (2015), 237–249. https://doi.org/10.1016/j.micpro.2015.04.007

[15] Jin Sha, Jingbo Liu, Jun Lin, and Zhongfeng Wang. 2016. A Stage-Combined Belief Propagation Decoder for Polar Codes. *J. Signal Process. Syst.* (2016), 1–8. https://doi.org/10.1007/s11265-016-1181-y

[16] Cemaleddin Simsek and Kadir Turk. 2016. Simplified Early Stopping Criterion for Belief-Propagation Polar Code Decoders. *IEEE Commun. Lett.* 20, 8 (2016), 1515–1518. https://doi.org/10.1109/LCOMM.2016.2580514

[17] Bogdan Spinean, Georgi Kuzmanov, and Georgi Gaydadjiev. 2011. Vector processor customization for FFT. *Proc. - 2011 Int. Conf. Embed. Comput. Syst. Archit. Model. Simulation, IC-SAMOS 2011* (2011), 110–117. https://doi.org/10.1109/SAMOS.2011.6045451

[18] Shuanghong Sun and Zhengya Zhang. 2016. Architecture and optimization of high-throughput belief propagation decoding of polar codes. In *2016 IEEE Int. Symp. Circuits Syst.*, Vol. 2016-July. IEEE, 165–168. https://doi.org/10.1109/ISCAS.2016.7527196

[19] Ido Tal and Alexander Vardy. 2015. List Decoding of Polar Codes. *IEEE Trans. Inf. Theory* 61, 5 (may 2015), 1–11. https://doi.org/10.1109/TIT.2015.2410251 arXiv:1206.0050

[20] Menghui Xu, Shusen Jing, Jun Lin, Weikang Qian, Zaichen Zhang, Xiaohu You, and Chuan Zhang. 2018. Approximate Belief Propagation Decoder for Polar Codes. In *2018 IEEE Int. Conf. Acoust. Speech Signal Process.* IEEE, 1169–1173. https://doi.org/10.1109/ICASSP.2018.8462478

[21] Kai Zhang, Shuming Chen, Sheng Liu, Yaohua Wang, and Junhui Huang. 2011. Accelerating the data shuffle operations for FFT algorithms on SIMD DSPs. *Proc. Int. Conf. ASIC* (2011), 683–686. https://doi.org/10.1109/ASICON.2011.6157297