Architecture and Optimization of High-Throughput Belief Propagation Decoding of Polar Codes

Shuanghong Sun and Zhengya Zhang
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI, 48109-2122

***Abstract*—Belief propagation (BP) is one common decoding algorithm for polar codes. BP can be implemented using a forward-backward flooding schedule that removes the data dependency. The intrinsic parallelism of the BP algorithm permits a high throughput. In this paper, we present a parallel BP decoder design space exploration. Through gate-level implementations, we analyze the silicon area, throughput, and power consumption of each design. We present adaptive quantization and early convergence detection to improve the error-correcting performance and throughput of BP decoders.**

I. Introduction

Polar codes have received much attention recently because it is provably Shannon limit achieving, and it has a low decoding complexity and a high scalability [1]. The common decoding algorithms of polar codes are successive cancellation (SC) [1] and belief propagation (BP) [2]. A BP decoder provides a high throughput, but the error-correcting performance of a BP decoder is slightly worse [3] than an SC decoder. A BP decoder also requires a larger memory for implementation.

Variations of SC decoding have been proposed to improve the throughput and error-correcting performance of polar codes. SC list [4] is a powerful decoding algorithm, but it increases decoding complexity. Simplified SC (SSC) [5] and SSC with maximum-likelihood (SSC-ML) [6] improve the decoding throughput, but the design is dependent on the bit selection, so the architecture is less scalable. BP decoding can also be improved to close the gap with SC decoding, e.g., through bit selection or by concatenating a polar code with an outer code [7].

In this paper, we focus on the design of high performance BP decoders. By constructing and analyzing a family of BP decoding architectures, we demonstrate the flexibility of the design along with optimization techniques.

II. Background

For a block length $N = 2^n$ polar code, its generator matrix G is the n-th Kronecker power of matrix $F = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$, i.e., $G_N = F^{\otimes n}$ [1]. Encoding is done by $x = uG$, where u is the binary input vector and x is the codeword. x is modulated and sent to the channel. At the output of the channel, soft vector y is received. A polar decoder attempts to recover u from y.

Fig. 1 is the factor graph of the polar code of block length $N = 8$, with bit-reversal shuffled inputs [2]. The graph consists of $\log_2 N$ columns and each column is made up of $N/2$ nodes. Shuffling of inputs allows the input and output connections of the nodes in every column to be same. When the factor graph is used for encoding, the plus sign in the graph represents an XOR operation, and the equal sign represents a pass-through.

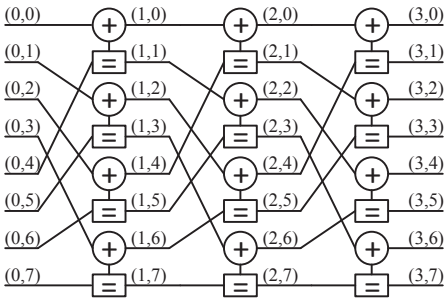

Fig. 1. Factor graph of an 8-bit polar code.

When the codeword is sufficiently long, the capacity of each bit approaches either 1 or 0. Highly reliable bits are used to carry information and the unreliable bits are frozen to a known value, normally 0. $\mathcal{A}$ denotes the set of frozen bits [1].

A. SC Decoding

In SC decoding, the message bits u_0 to u_{N-1} are decoded successively. If $i \in \mathcal{A}$, $\hat{u}_i = 0$; otherwise $\hat{u}_i$ is decoded by the maximum likelihood decision rule:

$$\hat{u}_i = \begin{cases} 0 & \text{if } \frac{P(y,\hat{u}_0^{i-1}|u_i=0)}{P(y,\hat{u}_0^{i-1}|u_i=1)} > 1 \\ 1 & \text{otherwise} \end{cases}$$

where $P(y, \hat{u}_0^{i-1}|u_i = b)$ refers to the probability that the received vector is y and the previously decoded bits being $\hat{u}_0$ through $\hat{u}_{i-1}$, given the current bit being b, $b \in \{0, 1\}$. The ratio of the probability given $u_i = 0$ over the probability given $u_i = 1$ is the likelihood ratio (LR) of bit u_i [8]. In designing an SC decoder, log likelihood ratio (LLR) is often used to simplify the implementation [8].

B. BP decoding

BP decoding operates on the factor graph by passing the frozen set information from left to right in a right propagation (R propagation), and passing the received channel outputs from right to left in a left propagation (L propagation).

When the factor graph is used for decoding, each node in the graph, shown in Fig. 2(a), is implemented using a f function that acts as a soft XOR (represented by the plus sign), and a summation (represented by the equal sign). The

978-1-4799-5341-7/16/$31.00 ©2016 IEEE

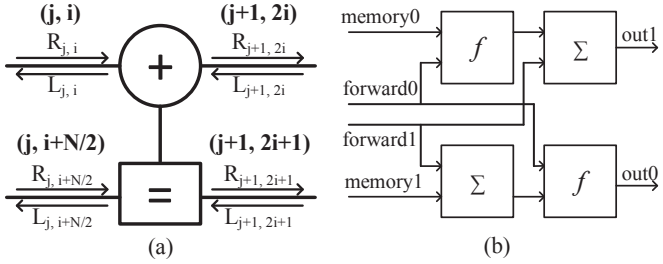

Fig. 2. (a) A node in the factor graph and (b) its functional block diagram.

computation performed by the node is described in equation set (1), where the notation $L_{j,i}$ ($R_{j,i}$) represents the left-bound message (right-bound message) at column j and row i of the factor graph and $f(a,b) = \text{sign}(a)\text{sign}(b)\min(|a|,|b|)$ [2].

$$\begin{aligned} L_{j,i} &= f(L_{j+1,2i+1} + R_{j,i+N/2},\ L_{j+1,2i}) \\ L_{j,i+N/2} &= f(L_{j+1,2i},\ R_{j,i}) + L_{j+1,2i+1} \\ R_{j+1,2i} &= f(L_{j+1,2i+1} + R_{j,i+N/2},\ R_{j,i}) \\ R_{j+1,2i+1} &= f(L_{j+1,2i},\ R_{j,i}) + R_{j,i+N/2} \end{aligned} \quad (1)$$

BP decoding is intrinsically column-wise parallel, i.e., all $N/2$ nodes in one column can be computed concurrently.

III. BP Decoder Architectures

A BP decoder is designed by mapping the factor graph. A decoding iteration starts with R propagation, followed by L propagation. At the end of each L propagation, the L messages from the final column of nodes are used as the posteriors for making hard decisions. Convergence usually requires between 5 and 10 iterations depending on signal-to-noise ratio (SNR).

Each node in the factor graph is implemented as a processing element (PE). The processing follows a read-compute-write 3-stage pipeline shown in Fig. 3(a). The first pipeline stage reads two L (R) messages from memory. The second pipeline stage takes additional two R (L) messages forwarded from pipeline registers and computes new messages following equation set (1). The final pipeline stage forwards the new messages to the next column of nodes via pipeline registers and stores a copy of the messages to memory. The memory blocks are usually merged to a large array for efficiency.

A. Architecture Building Blocks

A BP decoder is constructed using PEs to process messages and using memory and registers to store messages. PEs and memory elements are connected using muxes.

1) PE: Equation set (1) specifies that a PE computes two R messages and two L messages every time. However, in R propagation, the L messages computed by the PEs are overwritten in the subsequent L propagation. Similarly, the R messages computed in L propagation are overwritten in the subsequent R propagation. These redundant operations can be eliminated, resulting in a PE that computes only R messages in R propagation and only L messages in L propagation.

The calculation of R messages in R propagation and the calculation of L messages in L propagation are identical. Therefore, a PE only requires one set of hardware that is made up of two f operators to compute compare-select and two adders, as shown in Fig. 2(b).

2) Memory: One column of nodes produce N Q-bit new messages, where Q is the message word length that ranges from 4 to 6 bits, and the messages are saved in memory. Accounting for both R and L propagation, the total memory requirement is $2NQ\log_2 N$ to store all the R and L messages.

The memory access follows a well-defined pattern: in R propagation, L messages are consumed, and their places in memory can be replaced by newly produced R messages, and vice versa. As a result, the memory size can be reduced by half to $NQ\log_2 N$. Note that the output messages of the final column of nodes do not need to be saved, so the memory size is further reduced to $NQ(\log_2 N - 1)$.

The memory can be implemented in SRAM arrays or register files. In a highly parallel architecture, the memory is implemented in register files to support wide parallel access by many PEs; in a less parallel architecture, the memory is implemented in SRAM to save area.

In a pipelined implementation, pipeline registers are inserted after the PE as output registers. The output messages of one column of nodes can be directly forwarded to the next column of nodes in high-throughput designs.

B. Single-Column Architecture

In a single-column architecture, shown in Fig. 3(b), $N/2$ PEs are used to process a column of the factor graph at a time. Message memory is organized to have $\log_2 N - 1$ words and NQ bits in each word to support reading and writing of N Q-bit messages by the $N/2$ PEs in one clock cycle.

Using a single-column architecture, one decoding iteration takes $2\log_2 N$ clock cycles: $\log_2 N - 1$ cycles are spent on R propagation (going from column 1 through $\log_2 N - 1$), and $\log_2 N$ cycles are spent on L propagation (going from column $\log_2 N$ to 1) and 1 pipeline filling cycle between R and L propagation. The clock period is determined by input routing (including mux and wire delay), PE processing, output routing, and pipeline register setup time.

Due to the regularity of polar code structure, i.e., the factor graph of a shorter code is a sub-graph of that of longer codes, decoder architecture can be folded or unfolded to meet different application specifications. A double-column architecture and a half-column architecture are described below as examples of architectural transformations.

C. Double-Column Architecture

A double-column architecture is constructed by unfolding the single-column architecture by 2 [9], as shown in Fig. 3(c). The double-column architecture uses N PEs to process two consecutive columns of the factor graph in one cycle. The N PEs are separated to two sections, with section$_{odd}$ processing an odd-numbered column and section$_{even}$ processing an even-numbered column. The outputs of section$_{odd}$ are directly routed to section$_{even}$. The memory is also divided into two

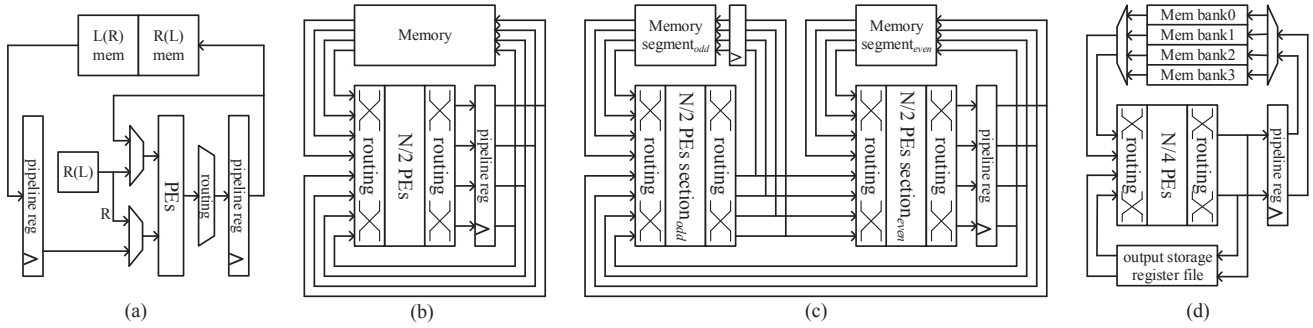

Fig. 3. (a) Pipelined datapath of a BP decoder and architecture of (b) single-column (c) double-column and (d) half-column BP decoder.

segments, and each segment is dedicated to a PE section, storing either the messages from the odd-numbered columns or the even-numbered columns. Segment$_{odd}$ stores $\lceil(\log_2 N-1)/2\rceil$ words and segment$_{even}$ stores $\lfloor(\log_2 N-1)/2\rfloor$ words, with NQ bits in each word. The total size of the memory remains unchanged compared to the single-column architecture.

The double-column architecture shortens the number of cycles per decoding iteration to $\log_2 N+2$, but the clock period is longer. Compared to the single-column architecture, this architecture adds a second section of PE processing and routing. Due to the merged routing from section$_{odd}$ to section$_{even}$, the increase in clock period is usually much less than 100% [9], resulting in a net decrease of decoding latency and increase of throughput.

D. Half-Column Architecture

A half-column architecture is constructed by folding the single-column architecture by a factor of 2 shown in Fig. 3(d). The half-column architecture uses $N/4$ PEs to process either the top or the bottom half of a column of the factor graph.

Using the 8-bit polar code shown in Fig. 1 as an example, a half-column architecture uses 2 PEs to process 2 nodes of a column at a time. Note that the top two nodes of column i take the R message inputs from bits $\{0,1\}$ and $\{4,5\}$ of column $i-1$, and the L message inputs from bits $\{0,1\}$ and $\{2,3\}$ of column i; the bottom two nodes take the R inputs from bits $\{2,3\}$ and $\{6,7\}$ and the L inputs from bits $\{4,5\}$ and $\{6,7\}$. The memory access of the half-column architecture can be generalized to a N-bit polar code. The architecture requires the message memory to be divided to four banks with each bank storing a quarter section of the bits. The total message memory size remains the same as the single-column architecture.

Since two cycles are needed to process one column of nodes, the outputs cannot be immediately forwarded to the next column processing and extra buffering is needed. Therefore, a half-column architecture requires an extra storage register file.

The half-column architecture uses half as many PEs as a single-column architecture, but the memory size is the same and it uses more registers, so the area of the decoder is only marginally reduced. The extra muxing overhead results in a longer clock period than the single-column architecture and the latency in terms of clock cycles is doubled.

TABLE I
SYNTHESIZED DECODER DESIGNS

	area (mm^2)	frequency (MHz)	power (mW)	throughput (Gbit/s) itr=5	energy (pJ/bit)
Single column	1.325	625	339	6.40	53.0
	1.404	769	442	7.87	56.2
Double column	2.033	400	359	6.83	52.6
	2.136	476	447	8.12	55.0
Half column	1.089	556	236	2.92	80.8
	1.152	714	318	3.75	84.8

E. Synthesis Results and Discussions

The single-column, double-column and half-column architectures are synthesized in a 65nm CMOS technology, and the results are listed in TABLE I for comparison. The architectures are designed for a $Q=6$ 1024-bit polar code, each of which is implemented in two designs that are presented in two rows: a low-power (LP) design and a high-performance (HP) design. The LP design is more compact and consumes lower power, but it sacrifices performance. The HP design uses upsizing and buffering, i.e., utilizes more area, to improve performance. In the following, we discuss the trade-off between the three architectures based on the LP designs. The HP designs follow the same trade-off.

The single-column LP design achieves a throughput of 6.4 Gb/s in decoding the 1024-bit polar code, consuming 53 pJ/b. The double-column LP design uses 53% more silicon area to increase the throughput to 6.83 Gb/s, consuming a lower energy of 52.6 pJ/b. Considering the large increase in area, the gain in throughput is relatively low, making the single-column architecture more advantageous, especially in terms of area efficiency.

The half-column LP architecture saves only 18% of the area, but the throughput drops to 2.92 Gb/s. The result is largely due to the constant memory size and the extra register file and muxing. The energy efficiency lowers to 80.8 pJ/b, making it less attractive than the single-column and double-column architectures. Folding the architecture further is unlikely to yield any improvement due to the constant memory size and the increased control overhead.

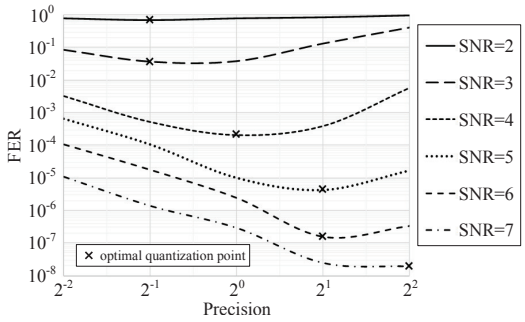

Fig. 4. Quantization effect on FER of a $Q = 6$ (1024, 512) code.

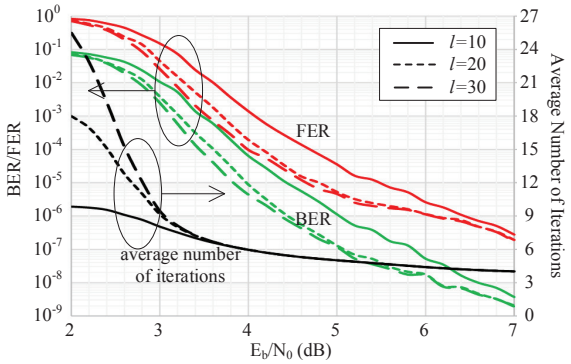

Fig. 5. Effect of maximum number of iteration on performance and latency.

IV. Adaptive Quantization and Early Termination

Hardware decoders are almost always implemented in fixed-point using a short word length to reduce cost. However, a short word length also results in a precision lost or an overflow leading to a degradation of error-correcting performance.

Polar codes do not have parity checks to indicate whether the decoding has converged, and more BP iterations are usually employed to improve performance. In most cases, convergence happens quickly and a large number of iterations is wasteful.

A. Adaptive Quantization

How to allocate the finite number of bits involves a trade-off between precision and range. For example, a word with a fixed length of $Q = 6$ can represent an integer in the range of $[-32, 31]$ with a precision of 2^0; or it can represent a smaller range of $[-8, 7.75]$ with a fine precision of 2^{-2}; or it can also represent a larger range of $[-128, 124]$ with a coarse precision of 2^2, i.e., the lower 2 bits are rounded off.

Fig. 4 shows that at a low SNR, a fine precision leads to a better frame error rate (FER) despite the lower range; while at a high SNR, when the mean and variance of the messages are larger, a higher range lowers FER. With a fixed word length, it is desirable to use an adaptive quantization depending on channel SNR. It can be observed that a fixed precision of 2^0 significantly degrades FER of this code at SNR (E_b/N_0) higher than 5 dB, which explains the flatter FER curves in high SNR range in Fig. 5. To implement adaptive quantization, only slicers and muxes are needed at the input to the decoder to select the desired quantization.

B. Early Termination

It is only possible to check the confidence in decoding polar codes for early termination. One early termination scheme is to check whether the hard decisions remain consistent for m consecutive iterations. Our experiments show that setting m to as low as 3 is sufficient in almost all cases to ensure a good error-correcting performance with negligible degradation.

With early termination, the decoder runs for at least m iterations in order to check convergence, so m sets the lower bound on the number of iterations. An upper bound, l, is set to prevent the decoder from being trapped in a case that never converges. Fig. 5 shows FER and BER of a $Q = 6$, precision $= 2^0$ (1024, 512) polar code, with early termination ($m = 3$). The FER and BER improve significantly when l increases from 10 to 20, but marginally from 20 to 30. For different l values, the average number of iterations falls below 6 for SNR higher than 4 dB, therefore the throughput becomes independent from l. To implement early termination, $2N$ XNORs and an N-input AND need to be added to detect convergence.

V. Conclusion

We present a family of BP decoder architectures for polar codes and their gate-level implementations to demonstrate the trade-offs between area, power, and throughput. The single-column architecture stands out as the most efficient when considering both area and energy consumption. The hardware decoders can adopt a simple SNR-adaptive quantization scheme to improve the error-correcting performance, and an early termination scheme based on consecutive hard decisions to reduce the average decoding latency.

Acknowledgment

This work was supported in part by Intel Corporation and NSF CCF-1054270. We would like to thank Dr. F. Sheikh and collaborators at Intel Labs for advice.

References

[1] 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, July 2009.

[2] A. Pamuk, "An FPGA implementation architecture for decoding of polar codes," in *Int. Symp. Wireless Commun. Syst.*, Nov 2011, pp. 437–441.

[3] B. Yuan and K. K. Parhi, "Architecture optimizations for BP polar decoders," in *IEEE Int. Conf. Acoustics, Speech and Signal Process.*, May 2013, pp. 2654–2658.

[4] I. Tal and A. Vardy, "List decoding of polar codes," in *IEEE Int. Symp. Inf. Theory*, July 2011, pp. 1–5.

[5] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Commun. Lett.*, vol. 15, no. 12, pp. 1378–1380, December 2011.

[6] G. Sarkis and W. J. Gross, "Increasing the throughput of polar decoders," *IEEE Commun. Lett.*, vol. 17, no. 4, pp. 725–728, April 2013.

[7] J. Guo, M. Qin, A. A. Guillen i Fabregas, and P. H. Siegel, "Enhanced belief propagation decoding of polar codes through concatenation," in *IEEE Int. Symp. Inf. Theory*, June 2014, pp. 2987–2991.

[8] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Trans. Signal Process.*, vol. 61, no. 2, pp. 289–299, Jan 2013.

[9] Y. S. Park, Y. Tao, S. Sun, and Z. Zhang, "A 4.68Gb/s belief propagation polar decoder with bit-splitting register file," in *Symp. VLSI Circuits Dig. Tech. Papers*, June 2014, pp. 1–2.