Parallel Decoding of Turbo Codes

Dejan Spasov(✉)

Faculty of Computer Science and Engineering, Skopje, Macedonia
dejan.spasov@finki.ukim.mk

Abstract. Given a turbo code generated by parallelly concatenated recursive systematic convolutional encoders, the turbo decoder comprises MAP decoders coupled in a serial connection, where each MAP decoder decodes a recursive systematic convolutional code. We propose a turbo decoding algorithm that, on a logical level of abstraction, is made of several turbo decoders working in parallel. Each turbo decoder is initialized with different recursive convolutional code. Practical implementation of the proposed algorithm may be achieved with a single turbo decoder, where MAP decoders are working concurrently.

Keywords: Turbo codes · MAP decoding · MAX-Log-MAP decoding
Turbo decoding · Convolutional codes

1 Introduction

Turbo Codes are a class of forward error correction codes invented by Berrou and first published in [1]. Turbo codes were the first practical system that achieved signal-to-noise ratio of 0.7 dB above the Shannon's limit while providing bit error probability of 10^{-5} [1]. Turbo codes are special sub-type of convolutional codes. In general, a turbo encoder is any combination of two or more identical convolutional encoders connected via interleavers. Traditionally, a turbo code comprises two recursive systematic convolutional (RSC) encoders coupled in a parallel concatenation scheme (Fig. 1). Turbo codes are systematic codes, which means that the input sequence appears unmodified at the output as sequence x. Figure 1 shows two recursive systematic convolutional encoders that output two coded sequences (x, y_1) and (x, y_2), where sequences y_1 and y_2 represent the parity bits. The turbo code on Fig. 1 initially has a code rate of 1/3; however, higher code rates may be achieved by applying various puncturing patterns. An example of puncturing pattern may be alternating between the parity bits y_1 and y_2. The interleaver is a device that outputs a random permutation of the received sequence. By providing random permutation of the input sequence, the interleaver allows identical recursive encoders to be used in the hardware design. Thus, two identical recursive systematic convolutional codes coupled with a random interleaver behave as two different recursive systematic convolutional codes.

Decoding of a recursive systematic convolutional code may be done with the Viterbi algorithm [2] or with Maximum A Posteriori Probability (MAP) algorithm [3]. Decoding of the turbo codes, which are made of two parallel systematic recursive convolutional codes, involves separate decoding of each of the systematic recursive convolutional codes (Fig. 2).

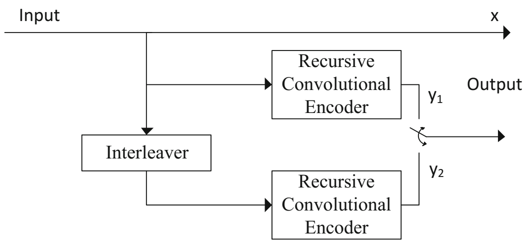
Input
x
Recursive Convolutional Encoder
y1
Output
Interleaver
y2
Recursive Convolutional Encoder

Fig. 1. Turbo encoder

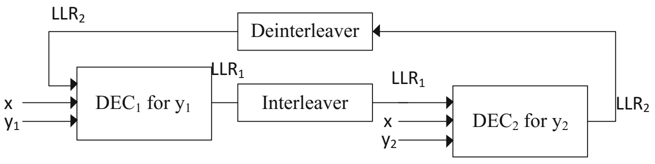

Fig. 2. Turbo decoder

The turbo decoder (Fig. 2) is made of two decoders DEC$_1$ and DEC$_2$, which are modified MAP decoders, also known as BCJR decoders [1]. The two decoders DEC$_1$ and DEC$_2$ may share information several times before the turbo decoder outputs estimates for each bit. This information sharing is known as iteration. Thus, the turbo decoder performs several iterations before outputting decision for each bit. The turbo decoder may be configured to perform two iterations simultaneously [4]. For example, when DEC$_1$ is working on the i-th iteration, DEC$_2$ may be working on the (i−1)-th iteration. The two decoders DEC$_1$ and DEC$_2$ may be Max-Log-Map decoders [6, 7], which is a simplified variant of the MAP decoder that involves the Viterbi algorithm [2]. More on implementation issues may be found in [5, 8]. Decoders DEC$_1$ and DEC$_2$ are configured to output the logarithm of likelihood ratio (LLR) for each bit x_k

$$LLR(x_k) = \log \frac{\Pr\{x_k = 1|observation\}}{\Pr\{x_k = 0|observation\}} \tag{1}$$

where $\Pr\{x_k = 1\, or\, 0|observation\}$ is a posteriori probability of the bit x_k. The decoder DEC$_1$ is activated first and it decodes the encoded sequence (x, y_1) and outputs LLR$_1$ quantities for each bit x_k. The LLR$_1$ quantities are then fed to DEC$_2$ that decodes the encoded sequence (x, y_2) to produce its own estimates LLR$_2$. The turbo decoding process continues in iterative fashion, and in the next iteration LLR$_2$ quantities are fed into DEC$_1$.

From Fig. 2, it may be observed that the turbo decoder is initialized with the first recursive convolutional code (x, y_1). In this paper we propose a turbo decoding scheme made of two serial turbo decoders that operate in parallel. The first turbo decoder is initialized with the first recursive convolutional code (x, y_1) and the second turbo decoder is initialized with the second recursive convolutional code (x, y_2). Results from both decoders are combined to produce the logarithm of likelihood ratio (LLR) for each bit x_k.

2 The MAP Decoding Algorithm

A convolutional encoder with M registers is finite state machine with 2^M states. Trellis diagram is labelled n-partite graph, in which every path represents a valid codeword (Fig. 3). Vertices of the $n+1$ disjoint sets in the trellis represent all possible 2^M states of the encoder. Vertices are labelled as decimal numbers, such that the content of the leftmost register corresponds to the most significant bit in the decimal number. Edge labels represent the input letters to the encoder and the appropriate output letters produced by the encoder separated by the slash symbol.

Trellis diagram of convolutional codes gives a hint about the decoding process; if the received sequence does not represent a valid path through the trellis diagram, then we can conclude that errors have occurred. The decoding objective is to find the most probable valid path though the trellis. Several decoding algorithms exist for decoding convolutional codes. The most famous are the Viterbi algorithm [2] and the BCJR algorithm [3]. The Viterbi algorithm is universally used and is highly parallelizable.

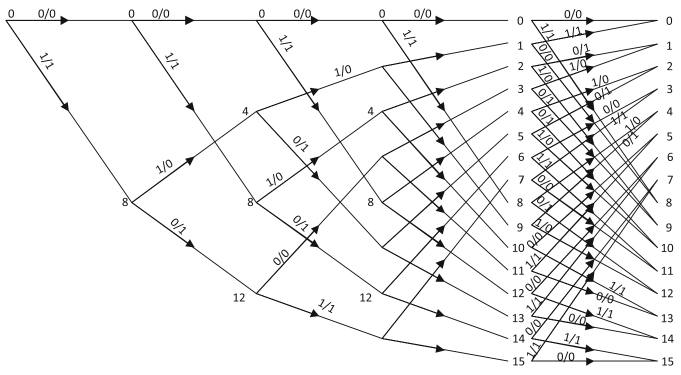

Fig. 3. Trellis diagram of a convolutional code

The BCJR algorithm can be envisioned as two stage process. In the first stage, known as *the forward α recursion*, the decoder moves through the trellis in left to right fashion and with each state s it associates a probability function $\alpha_i(s)$ that is recurrently computed. In the second stage, known as the *backward β recursion*, the BCJR decoder recurrently computes additional probability function $\beta_i(s)$ and then using the stored $\alpha_i(s)$ outputs the logarithm of likelihood ratio for each bit x_k.

Let $\alpha_i(s), s = 0, 1, \ldots, M-1$, be a set of state metrics on the n-partite trellis at the time i. The computation of the forward α probabilities starts from the initial conditions

$$\begin{cases} \alpha_0(s) = 1 & s = 0 \\ \alpha_0(s) = 0 & s \neq 0 \end{cases} \tag{2}$$

and following the edges of the trellis with non-zero branch probabilities $\gamma_i(s, s')$, the decoder, at each iteration stores all $\alpha_i(s)$ and computes $\alpha_{i+1}(s)$, according to

$$\alpha_{i+1}(s') = \sum\nolimits_{s=0}^{M-1} \alpha_i(s)\gamma_i(s, s'), \tag{3}$$

where $s' = 0, 1, \ldots, M - 1$. Stored $\alpha_i(s)$ are used during the backward computation in order to compute *the log-likelihood-ratio* (1) for the i-th information bit x_i.

Let $\beta_i(s), s = 0, 1, \ldots, M - 1$, be another set of state metrics on the n-partite trellis at the time i. The computation of the backward β probabilities starts from the initial conditions $\beta_{N-1}(s) = \frac{1}{M}$ and following the edges of the trellis with non-zero branch probabilities $\gamma_i(s, s')$, the decoder, at each iteration computes $\beta_i(s)$, according to

$$\beta_i(s') = \sum\nolimits_{s=0}^{M-1} \beta_{i+1}(s)\gamma_i(s, s'), \tag{4}$$

where $s' = 0, 1, \ldots, M - 1$. Then the logarithm of likelihood ratio (LLR) for each bit x_k is computed as

$$LLR(x_k) = Log\left(\frac{\sum_s \alpha_i(s) \sum_{s'} \gamma_i(s, s')\beta_{i+1}(s')}{\sum_s \alpha_i(s) \sum_{s'} \gamma_i^{-1}(s, s')\beta_{i+1}(s')}\right) \tag{5}$$

3 Design of Parallel Turbo Decoder

Figure 4 shows a block diagram of a turbo decoder. The turbo decoder is coupled to receive channel information of a turbo code. The received turbo code is made of two parallel recursive systematic convolutional codes RSC1 and RSC2. Principle of operation of the turbo decoder is described on Fig. 2. The turbo decoder is configured first to decode the first code RSC1, then the code RSC2. The decoder repeats this sequence for predefined number of iterations.

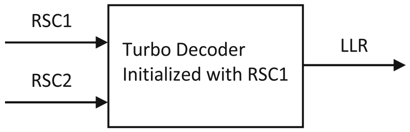

Fig. 4. Block diagram of a turbo decoder initialized with RSC1

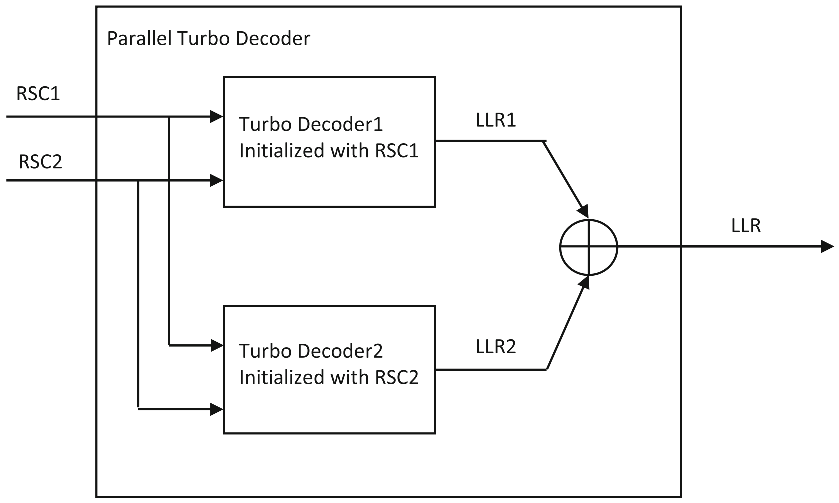

Fig. 5. Parallel turbo decoder

Parallel decoding of the turbo code made of two codes RSC1 and RSC2 may be achieved with two serial turbo decoders (as in Fig. 4) working in parallel, such that each turbo decoder is initialized with different RSC code (Fig. 5).

The parallel turbo decoder (Fig. 5) is coupled to receive channel information for two parallel recursive systematic convolutional codes RSC1 and RSC2. The parallel turbo decoder is made of two serial turbo decoders Turbo Decoder1 and Turbo Decoder2. Each of the serial turbo decoders start operation with different RSC code. The logarithm of likelihood ratio (LLR) for each bit x_k on the output of the parallel decoder is sum of logarithm of likelihood ratio (LLR) for each bit x_k on the output of each serial decoder. In general, the parallel turbo decoder may be generalized for any turbo code made of arbitrary number of recursive systematic convolutional codes in parallel connection. In practice, the parallel turbo decoder (Fig. 5) may be implemented with one serial turbo decoder (Fig. 2), where, at any moment, one MAP decoder computes the LLR1 coefficients from the Turbo Decoder1 and the other MAP decoder computes LLR2 coefficients from the Turbo Decoder2, simultaneously.

4 Practical Results

In our simulation of parallel turbo decoder, we use turbo code made of two recursive systematic convolution codes (as in Fig. 1). The turbo code is with code rate 1/2, which means that one of the two RSC codes is punctured out at any bit interval. The convolutional encoders are recursive with 16 states. Encoded sequence is 1025 bits long. The Turbo code is sent over Gaussian channel. To store state and branch metric we use IEEE 754 double precision format. Results are shown on Fig. 6.

Figure 6 compares performance of a regular turbo decoder (Fig. 2) and parallel turbo decoder (Fig. 5). Both turbo decoders are set to perform 8 iterations before outputting the

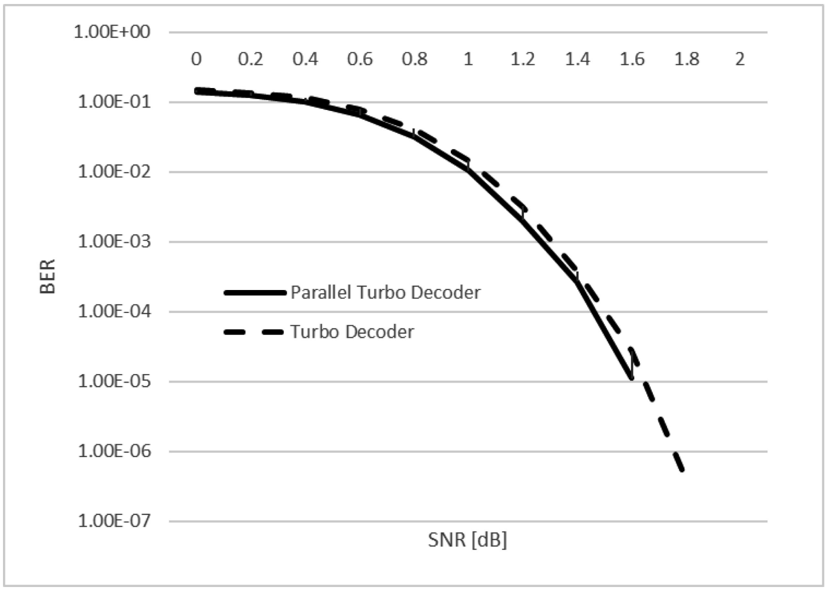

Fig. 6. Performance comparison between (regular) turbo decoder and parallel turbo decoder.

logarithm of likelihood ratio (LLR) for each bit x_k. Both the parallel turbo decoder (Fig. 5) and the regular turbo decoder (Fig. 2) are tested for bit error rate for various signal to noise ratios. Figure 6 confirms that the parallel turbo decoder (Fig. 5) performs better than the regular turbo decoder (Fig. 2), i.e. it shows improved bit error rate.

5 Conclusion

We have demonstrated a turbo decoding algorithm that improves bit error rate in decoding turbo codes by using parallel turbo decoders (Fig. 5). From the perspective of improved performance, it is obvious from Fig. 6 that the parallel turbo decoder improves the classical decoder. From the perspective of running time, using the same hardware, the parallel turbo decoding algorithm may be slower than the classical turbo decoding algorithm by a factor of two. However, if we can double the hardware resources, the parallel turbo decoding algorithm may be as fast as the classical turbo decoding algorithm.

Two lines of research may be identified in Turbo codes, with goals to develop a suboptimal turbo decoder that improves a certain bottleneck. One line of research is trying to minimize memory complexity; thus, requiring less die area and less power [6, 7]. Another line of research is to improve decoding speed of turbo decoders and to achieve the decoding speed of the LDPC codes. In this line of research, the trellis diagram of the turbo code (Fig. 3) is divided in subtrellises, which are decoded in parallel [9]. Advantage of the parallel turbo decoder (Fig. 5) that is introduced in this paper is that it can be applied on any suboptimal turbo decoder.

Acknowledgement. This work was partially financed by a project from the Faculty of Computer Science and Engineering at the Ss. Cyril and Methodius University, Skopje, Macedonia.

References

1. Berrou, C., Glavieux, A., Thitimajshima, P.: Near Shannon limit error-correcting coding and decoding: Turbo-codes. In: Proceedings of the ICC, Geneva, Switzerland, May 1993
2. Forney, G.D.: The viterbi algorithm. Proc. IEEE **61**(3), 268–278 (1973)
3. Bahl, L.R., Cocke, J., Jelinek, F., Raviv, J.: Optimal decoding of linear codes for minimizing symbol error rate. IEEE Trans. Inf. Theory **20**(3), 284–287 (1974)
4. Hagenauer, J., Offer, E., Papke, L.: Iterative decoding of binary block and convolutional codes. IEEE Trans. Inf. Theory **42**(2), 429–445 (1996)
5. Boutillon, E., Gross, W.J., Gulak, P.G.: VLSI architectures for the MAP algorithm. IEEE Trans. Commun. **51**(2), 175–185 (2003)
6. Zhan, M., Zhou, L.: A memory reduced decoding scheme for double binary convolutional turbo code based on forward recalculation. In: 7th International Symposium on Turbo Codes and Iterative Information Processing (ISTC), Gothenburg, Sweden (2012)
7. Choi, H.-M., Kim, J.-H., Park, I.-C.: Low-power hybrid turbo decoding based on reverse calculation. In: ISCAS, pp. 2053–2056 (2006)
8. Boutillon, E., Douillard, C., Montorsi, G.: Iterative decoding of concatenated convolutional codes: implementation issues. Proc. IEEE **95**(6), 1201–1227 (2007)
9. Maunder, R.G.: A fully-parallel turbo decoding algorithm. IEEE Trans. Commun. **63**(8), 2762–2775 (2015)