Computation reduction for turbo decoding through window skipping

Computation reduction for turbo decoding through window-skipping

M. Martina, C. Condo, M. Ruo Roch, G. Masera

A simple and effective technique to skip the computation of reliable portions of a frame (windows) for turbo code decoding is proposed. The proposed criterion relies on a very simple approximation of cross-entropy measure by the means of thresholding. This criterion features negligible complexity and low memory requirements. Simulation results show that, in the best case, up to the 20% of windows can be skipped with no error-rate degradation. Such a significant computation reduction can be exploited to directly reduce the power consumption as well.

Introduction: Turbo codes are among the best-performing forward error correcting codes. Indeed, they have been adopted in several standards for wired and wireless communications including the Long-Term-Evolution (LTE) advanced standard for mobile communications. Since the decoding algorithm is iterative, high throughput can be achieved with hardware implementations. In most of the cases, parallel architectures, namely several processing elements (PEs) working concurrently, are employed. As a consequence, a large amount of data are concurrently read/written from/to the memory and processed. This produces huge bandwidth toward the memory and large switching activity inside the PEs. A first attempt to reduce the bandwidth during the writing operation has been proposed in [1], where the authors devise a mechanism to adaptively send data toward the memory only if the reliability of the information has increased during the last iteration. Unfortunately, this technique does not prevent large bandwidth during the reading operation and leads to some waste computation inside the PEs. Indeed, only when data are computed one can perform the reliability check and decide if the data will be sent to the memory of not. This work aims to make one step further, namely it proposes a technique to skip the computation of portions of data (windows), which are already reliable, with a sliding-window-based approach. The proposed technique, applied to the LTE turbo code decoder, saves up to the 20% of the computation (and bandwidth during both reading and writing operations) at 1.8 dB of Signal-to-Noise-Ratio (SNR), with no bit-error-rate (BER) performance loss. As long as one accepts BER performance degradation, the amount of saved computation increases as well.

Decoding algorithm: The iterative algorithm used in turbo code decoding is based on a Maximum-A-Posteriori (MAP) estimation, known as BCJR algorithm, which is performed in the logarithmic domain on Logarithmic-Likelihood-Ratios (LLRs), leading to the well-known Log-MAP algorithm. As convolutional turbo codes are the concatenation through an interleaver of two constituent convolutional codes, each iteration at the decoder side is made of two half iterations, one for each constituent code. Each constituent decoder, referred to as MAP or Soft-In-Soft-Out (SISO) decoder, works on the trellis representation of the constituent code. Let k be the k-th trellis step, each SISO decoder combines a-priori (λ_k^{apr}) and intrinsic (λ_k^{int}) LLRs to refine the reliability of each uncoded symbol. This operation is achieved by computing the extrinsic information $\lambda_k^{ext} = \lambda_k^{apo} - \lambda_k^{int,u} - \lambda_k^{apr}$, where $\lambda_k^{int,u}$ is the intrinsic systematic information for the uncoded symbol u and λ_k^{apo} is the a-posteriori information, which is computed as:

$$\lambda_k^{apo} = \max^*_{e:u(e)=u} \{b_k(e)\} - \max^*_{e:u(e)=\tilde{u}} \{b_k(e)\}, \qquad (1)$$

where e is one edge in the trellis, $u(e)$ is the systematic information of e and $\tilde{u}$ is one uncoded symbol taken as a reference (usually $\tilde{u}=0$). Each $b_k(e)$ term in (1) is obtained as $b_k(e) = \alpha_{k-1}[s^S(e)] + \gamma_k(e) + \beta_k[s^E(e)]$, where

$$\alpha_k[s] = \max^*_{e:s^E(e)=s} \left\{\alpha_{k-1}[s^S(e) + \gamma_k(e)\right\}, \qquad (2)$$

$$\beta_k[s] = \max^*_{e:s^S(e)=s} \left\{\beta_{k+1}[s^E(e) + \gamma_{k+1}(e)\right\}, \qquad (3)$$

$$\gamma_k(e) = \lambda_k^{int}(e) + \lambda_k^{apr}(e). \qquad (4)$$

Several approximations have been proposed in the literature for the implementation of the $\max^*$ operator. However, approximating the $\max^*$ operator with the simple $\max$ function (Max-Log-MAP) leads to minor BER performance loss, provided that a scaling factor (δ) is applied to λ_k^{ext} [2]. According with the notation shown in Fig. 1 (a) $s^S(e)$ and $s^E(e)$ represent the starting and ending states of e, respectively.

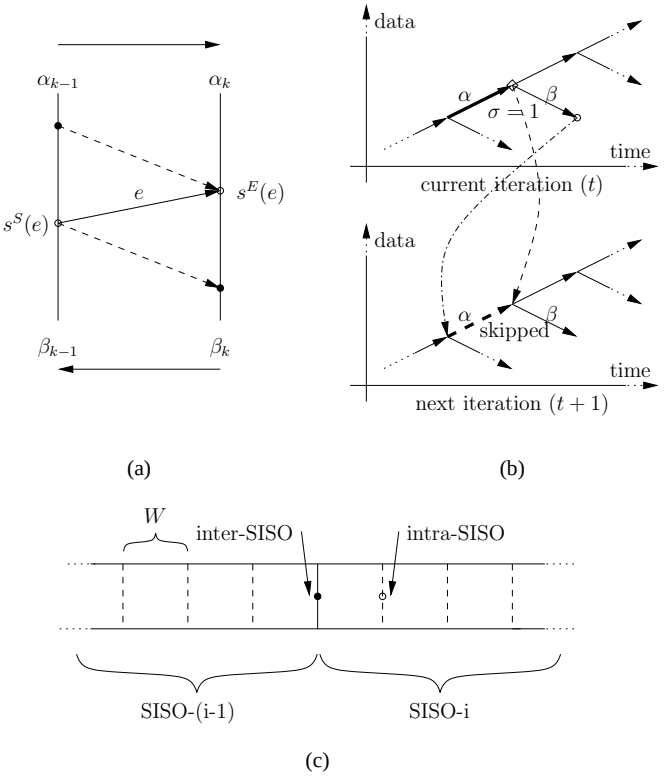

Fig. 1 *Used notation: trellis representation (a), border-metric-inheritance scheduling (b) and inter/intra-SISO window representation (c).*

Due to the recursion shown in (2) and (3), referred to as forward and backward recursion respectively, long frames would produce a large latency in the decoding. Thus, sliding-window-based decoding is routinely employed. A drawback of the sliding-window approach is the need for initializing border metrics in the backward recursion. A well-known solution to cope with this problem is the so-called β-training, where border metrics are estimated through a one-window training process. An effective alternative technique, referred to as border-metric-inheritance, has been proposed in [3] to avoid the use of β-training. Such a technique uses a memory to store border metrics computed during current iteration. These metrics will be used at the next iteration as the initialization values for the neighbouring windows. The corresponding scheduling is shown in Fig. 1 (b), where circle and diamond represent backward-border-metrics and forward-border-metrics, respectively. The same scheme can be extended to parallel decoders, where P SISO modules work concurrently on different portions of the trellis (see Fig. 1 (c)). Since the border-metric-inheritance technique causes a slight degradation of the BER performance, it is usually employed i) for the initialization of the windows in the backward recursion (backward-border-metric-inheritance), ii) for the initialization of inter-SISO windows, shown in Fig. 1 (c), where the first forward metric is inherited from the neighbouring SISO (inter-SISO forward-border-metric-inheritance). Forward border metrics for intra-SISO windows are already available from previous-window processing, so inheritance is not required.

Proposed technique: In [4] a bit level reliability criterion is proposed as a stopping rule for turbo code decoding. However, as shown in [5], bit level stopping comes at the expense of increasing the memory and the logic of each SISO module. This increase is caused by the data dependency in the BCJR algorithm. Indeed, due to the presence of the forward and backward recursions the computation of each λ_k^{ext} depends both on previous and successive metrics in the trellis. Since skipping steps in the trellis affects neighbouring LLRs, storing metrics is required to limit BER performance degradation. In [5] the convergence of each bit is monitored by studying the cross-entropy between a-priori and extrinsic information. This concept has been approximated in this current work by

applying a threshold to the magnitude of λ_k^{apr} and λ_k^{ext} in order to decide whether the computation of a window can be skipped or not. Skipping of reliable windows reduces the bandwidth toward the memory during reading operations as well.

Let N and W be the number of trellis steps in the whole frame and in one window, respectively. Thus, $M = N/P$ and $M_W = M/P$ are the number of trellis steps and the number of windows processed by each SISO module. Let $\lambda_{i,j,l}^{apr}$ and $\lambda_{i,j,l}^{ext}$ represent the a-priori and extrinsic LLRs at trellis step l into window j of SISO i, with $0 \leq i < P$, $0 \leq j < M_W$ and $0 \leq l < W$, namely $k = i \cdot M + j \cdot W + l$. For a given i, j couple the corresponding window is marked as to-be-skipped when $|\lambda_{i,j,l}^{apr}| \geq \theta$ and $|\lambda_{i,j,l}^{ext}| \geq \theta$ for every $l \in [0, W)$, where θ is a threshold. Let us assume that the comparison result is '1' when the comparison is true and '0' when it is false, then the skipping condition can be rewritten as

$$\sigma_{i,j} = \bigwedge_{l=0}^{W-1} (|\lambda_{i,j,l}^{apr}| \geq \theta) \wedge (|\lambda_{i,j,l}^{ext}| \geq \theta) \tag{5}$$

for window j in SISO i, where $\wedge$ is the `and` operation. Thus, when $\sigma_{i,j}$ is equal to one the window will be skipped and no further refined. However, this implies that forward and backward recursion on the neighbouring windows can not be initialized. This problem can be overcome by observing that with the border-metric-inheritance technique the effect of skipping a window is simply not updating the memory where border metric values are stored. As a consequence, both forward and backward recursions on the neighbouring windows can be initialized with the "old" α and β values, respectively. On the contrary, when $\sigma_{i,j}$ is equal to zero, forward border metrics for intra-SISO windows are available and forward-border-metric-inheritance is avoided. Let Q be the number of states of the code and $\alpha_{i,j+1,0}^{t+1}[s]$, with $s = 0, \cdots Q-1$, the first forward state metrics of intra-SISO window $j+1$ in SISO i at iteration $t+1$, then the proposed technique can be described as follows:

$$\alpha_{i,j+1,0}^{t+1} = \begin{cases} \alpha_{i,j,W}^{t} & \text{if } \sigma_{i,j}^{t} = 1 \\ \alpha_{i,j,W}^{t+1} & \text{otherwise} \end{cases}, \tag{6}$$

where $\sigma_{i,j}^{t}$ is the skipping condition defined in (5) at iteration t. As it can be observed, backward-border-metric-inheritance is used independently of the proposed technique. On the contrary, forward-border-metric-inheritance for intra-SISO windows is actually needed only when $\sigma_{i,j} = 1$. As a consequence, such a solution requires further memory to store the forward-border-metrics. Let us assume that n_s bits are used to represent each state metric, then $Q \cdot n_s \cdot M_W \cdot P$ bits are required to store all the forward-border-metrics. Since $\sigma_{i,j} = 1$ occurs when the decoder is converging, the following technique can be used instead: finding $\hat{s} = \max_s\{\alpha_{i,j,W}^{t}[s]\}$ and approximating each $\alpha_{i,j+1,0}^{t+1}[s]$ as

$$\alpha_{i,j+1,0}^{t+1}[s] = \begin{cases} 0 & \text{if } s = \hat{s} \\ -2^{n_s-1} & \text{otherwise} \end{cases}, \tag{7}$$

namely saturating the metrics. This approach requires to store $\hat{s}$, thus only $\log_2(Q) \cdot M_W \cdot P$ bits are needed. Moreover, as shown in the following section, this approximation does not lead to any BER performance degradation.

Experimental results: The proposed window-skipping technique has been applied to an LTE turbo code ($Q = 8$) decoder. The decoder uses the Max-Log-MAP algorithm with a scaling factor $\delta = 0.75$. The intrinsic and extrinsic information are represented with five and eight bits ($n_e = 8$) respectively and $n_s = 12$. The number of iterations has been fixed to eight for $N = 6144$, $P = 8$ and $W = 32$ and border-metric-inheritance is used for the backward recursion. As a consequence, the memory requirements are: 24.6 kbits for the state-metric memories [5] and 18.4 kbits for backward-border-metrics. Moreover, with the proposed technique, only 576 bits are needed for the approximated forward-border-metrics. Fig. 2 and Fig. 3 show the BER performance and the corresponding savings, which have been achieved for different values of $\theta = \lceil 2^{n_e-1}/\phi \rceil$. The value of θ is a fraction of the maximum value reached by $|\lambda^{apr}|$ and $|\lambda^{ext}|$, which is 2^{n_e-1}, over $\phi = 32, 16, 14, 12$, leading to $\theta = 4, 8, 10, 11$. As shown in Fig. 2 the proposed window skipping technique features no BER degradation for $\theta \geq 10$. In particular, with $\theta = 10$ the percentage of skipped windows increases from 5% at 1.2 dB up to 20% at 1.8 dB. However, reducing the constraint on the achieved BER further complexity can be saved. As an example for a target BER of 10^{-6} the proposed technique with $\theta = 8$ achieves more than the 23% of skipped windows. Similar results can be obtained with other codes and bit width.

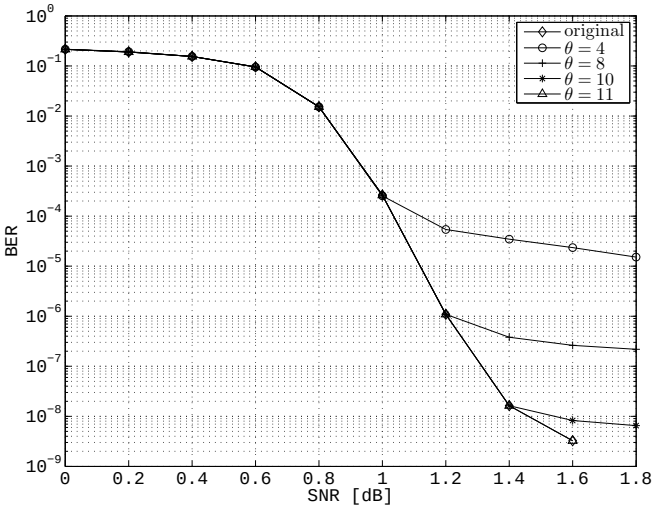

Fig. 2 *BER performance of the proposed window skipping technique for different values of θ.*

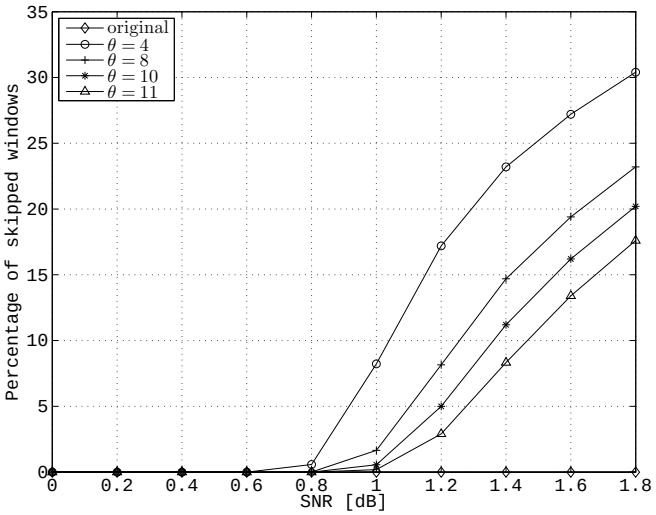

Fig. 3 *Percentage of saved windows of the proposed window skipping technique for different values of θ.*

Conclusions: In this work a window-skipping technique to reduce both the computation burden and the memory bandwidth in turbo code decoder architectures has been proposed. The proposed solution features extremely low memory requirements, being able to skip up to the 20% of windows with no BER degradation.

M. Martina, M. Ruo Roch and G. Masera (*Dipartimento di Elettronica e Telecomunicazioni - Politecnico di Torino - Italy*). Carlo Condo (*Department of Electrical and Computer Engineering, McGill University, Canada*)

E-mail: maurizio.martina@polito.it

References

1 O. Muller, A. Baghdadi, and M. Jezequel, "Bandwidth reduction of extrinsic information exchange in turbo decoding," *IET Electronics Letters*, vol. 42, no. 19, pp. 1104–1105, Sep 2006.

2 J. Vogt and A. Finger, "Improving the max-log-MAP turbo decoder," *IEE Electronics Letters*, vol. 36, no. 23, pp. 1937–1939, Nov 2000.

3 A. Abbasfar and K. Yao, "An efficient and practical architecture for high speed turbo decoders," in *IEEE Vehicular Technology Conference*, 2003, pp. 337–341.

4 D. H. Kim and S. W. Kim, "Bit-level stopping of turbo decoding," *IEEE Communications Letters*, vol. 10, no. 3, pp. 183–185, Mar 2006.

5 W. Shao and L. Brackenbury, "Early stopping turbo decoders: a high-throughput, low-energy bit-level approach and implementation," *IET Communications*, vol. 4, no. 17, pp. 2115–2124, 2010.