Advanced Iteration Overlap for Low-Latency Turbo Decoding

Jeremy Nadal, Stefan Weithoffer, Charbel Abdel Nour, Catherine Douillard

Advanced Iteration Overlap for Low-Latency Turbo Decoding

Jeremy Nadal, Stefan Weithoffer, Charbel Abdel Nour, Catherine Douillard
IMT Atlantique, Lab-STICC, UMR CNRS 6285, F-29238 Brest, France
e-mail: firstname.surname@imt-atlantique.fr

***Abstract*—Increasing the parallelism of a turbo decoder significantly constrains the inherently recursive decoding algorithm and limits its ability to efficiently achieve high decoding throughputs and to lower latency. This is partly due to the interleavers that impose precedence constraints when exchanging extrinsic information between the constituent decoders. In this paper, we introduce the notion of a generalized decoding schedule, which enables to configure the order in which decoder metrics and extrinsic information are exchanged during the decoding process. This schedule is generally inferred from the choices related to the hardware architecture and its parallelism degree. By following the proposed framework, it is possible to evaluate the achieved latency of any decoding schedule at a given operating point. We then propose a novel iteration overlap decoding schedule that can strike an excellent performance/latency tradeoff: Corresponding simulation results show that latency can be reduced by half for high rate LTE decoders for a given error rate target.**

***Keywords*—forward error correction, turbo decoder, shuffled decoding, low latency.**

I. Introduction

Turbo codes, a well-established class of iterative codes, are known for their inherent rate flexibility and low-complexity encoding. Over the span of more than 30 years since their introduction, they have been incorporated into several wireless communication standards such as Long Term Evolution (LTE) and Digital Video Broadcasting - Return Channel via Satellite (DVB-RCS/RCS2). Despite the preference for Low-Density Parity-Check (LDPC) codes in the 3rd Generation Partnership Project 5G New Radio (3GPP 5G NR) standard, turbo codes continue to play a role in the ongoing evolution of LTE [1], calling for an efficient increase in their decoding throughput.

With the emergence of Internet of Things (IoT) applications with low-latency requirements, the decoding latency represents an important performance metric [2]. The iterative nature of the turbo decoder penalizes latency. To address this issue, highly parallel hardware architectures have been proposed. Some architectures exploit *spatial parallelism* at the component decoder level, such as shuffled decoders which process component decoders in parallel with immediate extrinsic information exchange [3], [4] following their computation. Other architectures rely on *functional parallelism*, such as fully pipelining the decoding iterations, which achieves very high decoding throughput [5], [6].

The choice of the parallelism type and degree constrains the decoding schedule, i.e. the order in which metrics are computed, and the way extrinsic information is exchanged during the decoding process. These choices impact the convergence speed of the iterative decoder. Indeed, there is a trade-off between the latency reduction and the penalty in error correcting performance that has to be compensated for by additional decoding iterations (i.e. due to slower convergence of the iterative decoding process). Although this aspect is well-known in the literature, the convergence speed is generally studied for specifically designed hardware architecture types, and there is a lack of general decoding algorithms that can take into consideration arbitrary decoding schedules.

In this paper, we first propose a general formulation of the decoding algorithm where metrics and extrinsic information can be updated concurrently or sequentially in an arbitrary order. Through this novel formalism, the schedule becomes a full-fledged decoder parameter that can be optimized to minimize latency while meeting a target operating point defined by the error-rate and signal-to-noise ratio couple. We propose to restrict the schedule solutions to generalized schedules that are suitable for hardware implementation. In particular, the iteration overlap schedule introduced in [7] is extended to the case where interleaver precedence constraints are alleviated, allowing any overlapping depth.

The remainder of this paper is structured as follows: Section II briefly recalls background on parallel turbo decoders. In Section III, we introduce a general formulation of the decoding algorithm for arbitrary scheduling choices. Section IV presents the generalized XMAP schedule that includes the proposal of a novel iteration overlap schedule. Then, the corresponding latency savings are discussed in Section V and Section VI concludes the paper.

II. Background

The generalized process of turbo encoding consists in generating parity bits by providing K information (systematic) bits to N_{CD} constituent Recursive Systematic Convolutional (RSC) encoders with constraint length ν. For simplicity, we assume in this paper that all constituent encoders use the same RSC generator polynomial coefficients. The k^{th} input of the i^{th} constituent encoder is mapped to the $\Pi_i(k)^{\text{th}}$ information bit through interleaving function Π_i [1]. Once generated, the output bits are punctured to match the target code rate K/N, with N being the number of encoded bits. Formally, a turbo code can be defined by the tuple $\mathcal{C} = (\mathcal{I}, \mathcal{P}_i, \Pi_i)_{i\in[0,N_{\text{CD}}-1]}$,

[1]The first interleaver is usually the identity function: $\Pi_0(k) = k$

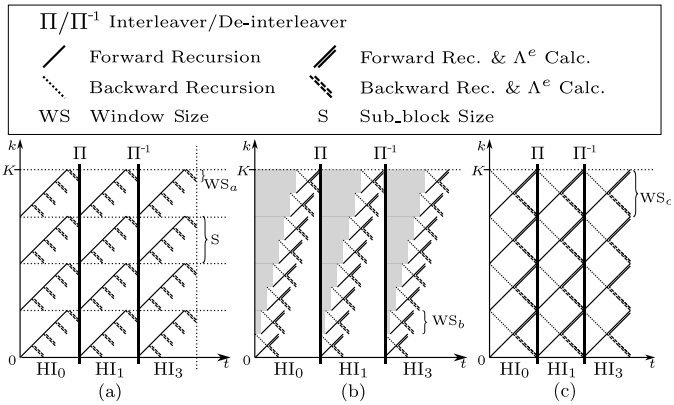

Figure 1: (a) Spatial parallelism (PMAP) (b) Functional parallelism (XMAP) and (c) Spatial and Functional parallelism (UXMAP).

where $\mathcal{P}_i$ denotes the list of punctured parity bit indices for the constituent code i and $\mathcal{I}$ the list of punctured systematic bit indices.

A. Decoding algorithm

A generalized turbo decoder comprises up to N_{CD} RSC Constituent Decoders (CDs), where extrinsic information on systematic bits $\Lambda_{e_i}, i \in [0, N_{\text{CD}} - 1]$, is exchanged in an iterative loop through the interleavers of $\mathcal{C}$. The complete execution of one constituent decoder corresponds to a *CD Processing* (CDP). During each CDP, the extrinsic information Λ_{e_i} is updated by recursively computing the forward/backward state metrics (α, β) from the first/last to the last/first trellis stages of the CD indexed by i.

Hardware architectures are typically designed for standardized codes such as the one in LTE [1], where the turbo decoder integrates $N_{CD} = 2$ CDs. To increase decoding throughput and reduce latency, K trellis stages are segmented into smaller *sub-blocks* and/or *windows* of size W_S. Then, each window/sub-block is processed using spatial or functional parallelism at the component decoder level.

B. Spatial parallelism

This type of parallelism is used in the Parallel Maximum A Posteriori (PMAP) decoder architecture [4], [8]. The decoding of the windows is performed in parallel on P serial *sub-decoder instances*. Figure 1 (a) shows the decoding schedule of a frame of size K with a PMAP architecture where $P = 4$ (implying a sub-block size $S = K/4$) with a window size of $W_{Sa} = S/4$.

In addition, several CDPs can be executed in parallel, then exchanging (partial) extrinsic information early, i.e. before the CDP completes. This refers to *shuffled decoding* [3]. The extreme case of a shuffled decoder where P is equal to the frame size K and $W_S = 1$ can be seen as the fully parallel MAP decoder proposed in [9].

C. Functional parallelism

This type of parallelism is used in the XMAP architecture [10], [11]. Several windows of size W_S are decoded in parallel while they are moving through the pipeline as illustrated in Fig. 1 (b) for the decoding of a frame of size K split into 4 windows of size W_S. The pipelining process can be extended by unrolling the iterative loop at the CD level. This leads to the Fully Pipelined Iteration Unrolled MAP (UXMAP) [5], [6] decoder architecture which integrates pipelined instances of several CDPs and windows (see Fig. 1(c)) such that complete frames are processed while moving through the pipeline.

It is important to note that functional and spatial parallelism-based architectures can be combined for achieving higher decoding throughputs, which leads to a very large design space of architectural choices. It is, for example, possible to process P XMAP cores in parallel, while the α/β-recursive operations are internally pipeplined in each XMAP.

D. Convergence speed and latency

The type of parallelism used in the decoder architecture determines the order in which the α/β-metrics are computed and extrinsic information are exchanged. Indeed, since these metrics are recursively computed in each CDP, the necessity of routing through interleaving functions impacts the number of metrics that can profit from an updated extrinsic information. This affects the convergence speed of the iterative decoder if the precedence constraints of the interleaver are not respected. For instance, dividing the frame into several windows or sub-blocks implies that the α/β-metrics are not immediately propagated to the edges of neighboring windows. Shuffled decoders noticeably suffer from this slower convergence since the precedence constraints of the interleaver are in general not respected for early exchange of partial extrinsic information: to benefit from full information exchanges between CDs, extrinsic information generated by the previous CDP has to be computed and exchanged before being *consumed* (i.e. used for computation) by the next CDP.

It is clear that there is a trade-off between the latency improvement achieved by the increased level of parallelism and the penalty in latency induced by the decreased convergence speed of the decoder since additional decoding iterations are required to meet a target error-rate performance. In the next section, we propose to formalize the turbo decoding algorithm in a more general way that takes into consideration that α/β-metrics and extrinsic information can be computed/exchanged in an arbitrary order.

III. Decoding schedule

A. Metrics and update of extrinsic information

An alternative way of describing the decoding process is to view metrics and extrinsic information as messages exchanged between different nodes, "*intra*" and "*inter*" CDPs respectively, in the graph representing the turbo decoder [12]. The nodes represent the systematic or parity bits and state variables for each CD. A node representing the code trellis constraints,

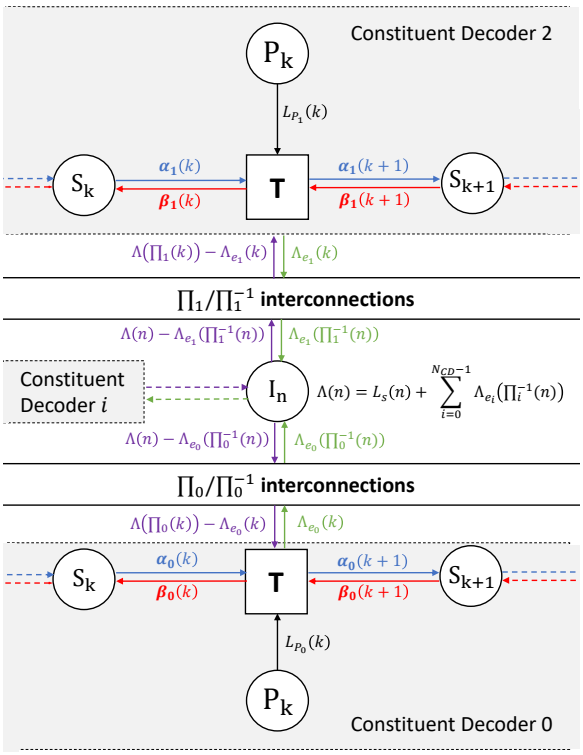

Figure 2: Graph representation of parallel turbo decoding.

identified as "*T-node*" in Fig. 2, is connected to state nodes ("*S-node*"), a parity bit node ("*P-node*") and a systematic bit node ("*I-node*"). This graph representation is well known in the literature [12] albeit not being the "standard" way to describe the decoding algorithm. Nonetheless, in the following, we will use it for clarity of representation to develop the way decoding and scheduling are performed.

In Fig. 2, the bottom and upper parts of the graph represent the trellis stages of CD 0 and 1 respectively. They are traversed in order to update extrinsic information $\Lambda_{e_0}(k)$ and $\Lambda_{e_1}(k)$ for systematic bits $\{\Pi_0(k), \Pi_1(k)\}$. The message transmitted from "*T-node*" k to "*S-node*" $k+1$ in CD i corresponds to the forward state metric vector $\boldsymbol{\alpha}_i(k+1)$ that stores the α-metrics for each of the 2^ν states, while the message transmitted from "*T-node*" k to "*S-node*" k is the backward state metric vector $\boldsymbol{\beta}_i(k)$ of CD i. Metrics and extrinsic information can be updated using the following set of equations

$$\boldsymbol{\alpha}_i(k+1) \leftarrow f_f\Big(\boldsymbol{\alpha}_i(k), \Lambda\big(\Pi_i(k)\big) - \Lambda_{e_i}(k), L_{P_i}(k)\Big) \quad (1)$$

$$\boldsymbol{\beta}_i(k) \leftarrow f_b\Big(\boldsymbol{\beta}_i(k+1), \Lambda\big(\Pi_i(k)\big) - \Lambda_{e_i}(k), L_{P_i}(k)\Big) \quad (2)$$

$$\Lambda_{e_i}(k) \leftarrow f_e\Big(\boldsymbol{\alpha}_i(k), \boldsymbol{\beta}_i(k+1), L_{P_i}(k)\Big) \quad (3)$$

where L_{P_i} are Log-Likehood Ratios (LLRs) of the received parity bits and (f_f, f_b, f_e) are the update functions currently used in typical decoding algorithms such as the well known log-MAP, max-Log-MAP (MLM) or local-SOVA algorithms [13], with or without quantization. In the above equations, the arrow "$\leftarrow$" symbol explicitly represents the update operation where the previously calculated values are replaced by the newly updated ones.

Instead of directly exchanging extrinsic information between CDs, it can be more convenient to update the soft output LLR of the systematic bits Λ by subtracting the relevant extrinsic information before update and adding the one obtained after update:

$$\Lambda(\Pi_i(k)) \leftarrow \Lambda(\Pi_i(k)) + f_e\Big(\boldsymbol{\alpha}_i(k), \boldsymbol{\beta}_i(k+1), L_{P_i}(k)\Big) - \Lambda_{e_i}(k), \quad (4)$$

with Λ having the channel LLRs of systematic bits L_S as initial values. Therefore, the extrinsic exchange process concurrently updates Λ and Λ_{e_i} through equations (4) and (3). Then, the decoding process consists in recursively updating the metrics and extrinsic messages through Eq. (1), (2), (3) and (4) several times. Note that the choice of the update order is flexible and, to the best of our knowledge, no efforts have been previously made to generalize and evaluate the impact of a chosen update order on the decoding process.

B. Decoding schedule

In most of prior art, the update of the different metrics follows the code trellis structure. Then, updated extrinsic information is exchanged with the next CD. For instance, updating $\boldsymbol{\alpha}_i(k)$ at a given processing time t followed by $\boldsymbol{\alpha}_i(k+1)$ at processing time $t+1$ ensures that $\boldsymbol{\alpha}_i(k+1)$ benefits from the updated extrinsic information acquired by $\boldsymbol{\alpha}_i(k)$. This successive (inherently serial) update at times $(t, t+1)$ results in a latency of 2 processing time slots. On the other hand, updating both $(\boldsymbol{\alpha}_i(k), \boldsymbol{\alpha}_i(k+1))$ concurrently (in parallel) at time t only takes 1 time slot, but the iterative process convergence rate can be penalized since $\boldsymbol{\alpha}_i(k+1)$ does not benefit anymore from the update of $\boldsymbol{\alpha}_i(k)$. For the sake of generality, it is then important to differentiate:

1) the code structure, which describes how metrics and extrinsic information are structurally dependent following the trellis and the interleaving function of systematic bits between CDs (graph structure in Fig. 2),
2) the order in which metrics and extrinsics are computed between exchanges, successively or concurrently.

The list of messages being updated at each time slot $t \in [0, T_L - 1]$, over a total of T_L time slots for the entire decoding process, is what defines a "*decoding schedule*". We qualify the schedule as "*static*" if the update list is predefined for a given code parametrization $\mathcal{C}$, i.e. independent of the reliability or varying states/values of the metrics during the decoding process. Otherwise, we define it as "*dynamic*". By this definition, iterative decoding with early stopping techniques [14] leads to dynamic schedules. [2]

[2]For clarity, this article focuses only on the static schedule types. However, the methods discussed in the following as well as the obtained schedules can still be used in combination with early stopping techniques (for example on-the-fly CRC calculation [15]) to maintain the associated latency reduction.

A static schedule can be formalized as $\mathcal{S} = (\mathcal{S}_0, ..., \mathcal{S}_t, ..., \mathcal{S}_{T_L-1})$, with $\mathcal{S}_t = \{\mathcal{S}_{f_i}(t), \mathcal{S}_{b_i}(t), \mathcal{S}_{e_i}(t)\}$ with $\{i = 0, ..., N_{CD} - 1\}$ where $\mathcal{S}_{f_i}(t), \mathcal{S}_{b_i}(t), (\mathcal{S}_{e_i}(t))$ represent the list of trellis stage indices of CD i where the update function of the α-metrics (Eq. (1)), β-metrics (Eq. (2)) and extrinsic information (Eq. (4) and Eq. (3)) must be executed concurrently at time slot t. For instance, a serial backward-forward decoding algorithm is obtained by configuring $\mathcal{S}$ such that

$$\mathcal{S}_{f_i}(t) = \{\text{mod}_K(t)\}, \tag{5}$$
$$\mathcal{S}_{b_i}(t) = \{\text{mod}_K(K - t - 1)\}, \tag{6}$$
$$\mathcal{S}_{e_i}(t) = \mathcal{S}_{f_i}(t) \cup \mathcal{S}_{b_i}(t), \text{if } \text{mod}_K(t) \geq K/2, \tag{7}$$

with $\ell = \lfloor t/K \rfloor$, $i = \text{mod}_{N_{\text{CD}}}(\ell)$ and $\mathcal{S}_{f_i}(t) = \mathcal{S}_{b_i}(t) = \mathcal{S}_{e_i}(t) = \emptyset$ if a value of t is not covered by the above set of equations. For instance, $\mathcal{S}_{f_i}(t) = \mathcal{S}_{f_i}(t) = \mathcal{S}_{e_i}(t) = \emptyset$ if $\text{mod}_{N_{\text{CD}}}(\lfloor t/K \rfloor) \neq i$.

C. Problem formulation

The turbo decoder can now be parameterized by the 2-tuple related to code structure and the schedule, namely $\mathcal{D} = (\mathcal{C}, \mathcal{S})$. For a particular code, to assess and compare the decoding performance of different parameter choices, two metrics are of particular interest: The error-rate performance metric $P_e(\mathcal{D}, \zeta)$ for a given signal-to-noise ratio value ζ and the latency of the decoding algorithm $T_L = \text{Card}(\mathcal{S})$. For the latter, it is assumed that the list $\mathcal{S}_t$ of metrics to be updated at time t corresponds to the same list of parallel updates performed by all hardware processing elements at clock cycle t. Furthermore, an additional constraint is put on the decoder schedule, where the maximum number of concurrent state metric and extrinsic information updates must be not exceed a given threshold $N_{\text{CU,target}}$ which reflects the available processing resources. Then, the decoding schedule $\mathcal{S}$ can be chosen to minimize decoder latency T_L while ensuring that an error-rate target $P_{e,\text{target}}$ (constraint $\mathscr{C}_1$) is respected within the available processing resources (constraint $\mathscr{C}_2$). This corresponds to the optimization problem

$$\mathscr{P}_{\mathcal{S}} : \underset{\mathcal{D}=(\mathcal{C},\mathcal{S})}{\text{minimize}} \Big(\text{Card}(\mathcal{S})\Big) \tag{8}$$
$$\text{s.t. } \mathscr{C}_1 : \; P_e(\mathcal{D}, \zeta) \leq P_{e,\text{target}}, \tag{9}$$
$$\mathscr{C}_2 : \; \max_t \Big(\sum_{x \in \{f_i, b_i, e_i\}_{\forall i}} \text{Card}\big(\mathcal{S}_x(t)\big)\Big) \leq N_{\text{CU,target}} \tag{10}$$

Alternatively, the inverse problem $\overline{\mathscr{P}}_S$ that consists in finding $\mathcal{S}$ which minimizes the error-rate metric under a maximum latency target $T_{L,\text{target}}$ can be defined as

$$\overline{\mathscr{P}}_S : \underset{\mathcal{D}=(\mathcal{C},\mathcal{S})}{\text{minimize}} \Big(P_e(\mathcal{D}, \zeta)\Big) \tag{11}$$
$$\text{s.t. } \mathscr{C}_2, \mathscr{C}_3 : \; \text{Card}(\mathcal{S}) \leq T_{L,\text{target}} \tag{12}$$

Note, however, that in this work we are focusing on the evaluation of solutions related to problem $\mathscr{P}_S$.[3]

IV. Advanced iteration overlap techniques

The task of finding an optimal solution to this non-linear problem is extremely challenging due to the intractable number of possible order combinations for the updates. Additionally, Monte Carlo simulations must be conducted to confirm that $P_e(\mathcal{D}, \zeta) \leq P_{e,\text{target}}$. Consequently, we limit the schedule choices discussed in the following to a subset of solutions suited for the XMAP architecture. Note, however, that the approach extends to other architectures as well.

A. Generic XMAP schedule

This section proposes a generalized XMAP schedule illustrated in Fig. 3 through an example with $N_{\text{CD}} = 2$. The processing of the first CD (index $i = 0$) starts at $t = 0$. All trellis stages are updated through $N_W = \lceil K/W_S \rceil$ XMAP windows for a total duration of T_{CD} time slots, with W_S denoting the window size. The start of CDP ℓ is offset from CDP $\ell-1$ by $\boldsymbol{T_P}[\ell]$ time steps where $\boldsymbol{T_P}$ is a vector containing the differential starting times between successive CDP. The resulting total latency of the decoding T_L is

$$T_L = T_{\text{init}} + \sum_{\ell=0}^{\ell_{\text{total}}-2} \boldsymbol{T_P}[\ell] + \underbrace{T_{\text{last}}}_{=T_{\text{CD}}} \tag{13}$$

where T_{init} is a fixed initial latency of the decoder due to pre-processing such as the initial β-metrics propagation for termination bit stages[4], T_{last} is the latency of the last CDP and ℓ_{total} is the number of performed CDPs during the decoding.

Within each CDP, the processing of each window $w \in [0, N_W - 1]$ starts after an intra CDP offset of $\boldsymbol{S}_\ell[w]$ time slots which reflects the window delays, where $\boldsymbol{S}_\ell$ is a N_w-length integer valued vector, which we introduced as "*window schedule*" in [7]. Since the underlying processing resources are the same for all CDPs, all window schedules $\boldsymbol{S}_\ell$ can be obtained through a permutation function p_ℓ such that $\boldsymbol{S}_\ell[k] = \boldsymbol{S}_{\ell+1}[p_\ell(k)]$. Consequently, the maximum window delay is the same for all CDPs ($\forall \ell, \; \max(\boldsymbol{S}_0) = \max(\boldsymbol{S}_\ell)$), and processing a complete CD always takes $T_{\text{CD}} = W_S + \max(\boldsymbol{S}_0)$ time slots.

The k^{th} extrinsic bit is updated at time slot $G_{e_i}(k)$ given by

$$G_{e_i}(k, \ell) = \lceil W_H \rceil + \underbrace{\Big\lfloor \Big| \text{ mod}(k, W_S) - W_H \Big| \Big\rfloor}_{\triangleq G_{\text{UX}}(k)} + \boldsymbol{S}_\ell[w_G(k)], \tag{14}$$

with $W_H = (W_S - 1)/2$ and $w_G(k) = \lfloor k/W_S \rfloor$. Note that $G_{\text{UX}}(k) = \lfloor | \text{ mod}(k, W_S) - W_H \; | \rfloor$ describes the

[3]Generally, practical applications have to reach a certain quality of service for an operating point defined by the couple $(P_{e,\text{target}}, \zeta)$. For instance, it is acknowledged that the Frame-Error-Rate (FER) target must be within 10^{-3} down to 10^{-5} when evaluating 4G-LTE or 5G systems [16]. On the other hand, for non-standard applications, $\overline{\mathscr{P}}_S$ may be considered.

[4]Termination bits used in 4G/LTE standard do not benefit from turbo exchanges, since their β-metrics never change during the decoding process.

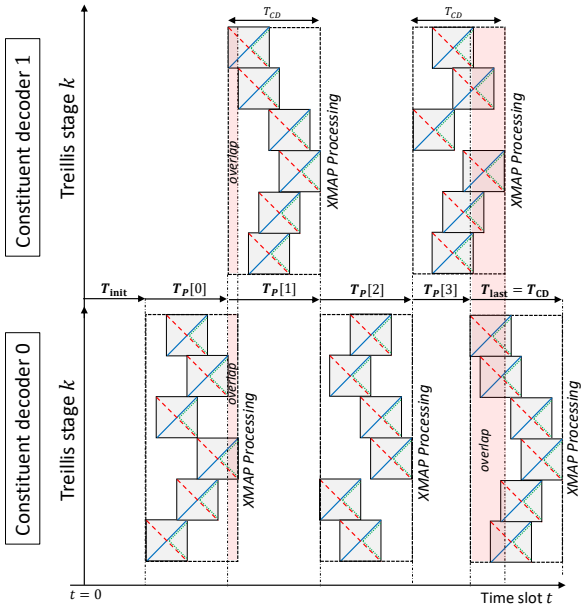

Figure 3: Generic XMAP scheduling for $N_{\text{CD}} = 2$.

generation times for the UXMAP case with full Iteration OverLap (IOL) [7]. Similarly, the generation time slots for the α and β metric updates can be expressed as $G_{f_i}(k,\ell) = \text{mod}(k, W_S) + \boldsymbol{S}_\ell[w_G(k)]$, and $G_{b_i}(k,\ell) = W_S - 1 - \text{mod}(k, W_S) + \boldsymbol{S}_\ell[w_G(k)]$, respectively [7]. Then, the complete decoding schedule $\mathcal{S}$ consists of the list containing the processing time slots for state metric or extrinsic information updates, computed for each trellis stage k within each of the ℓ CDPs. Each element of the list is computed starting from the generation time $G_{x_i}(k,\ell)$ within the current CDP added to the accumulated latency $\sum_{l=0}^{\ell-1} \boldsymbol{T_P}[l]$ stemming from all previous CDP computations:

$$k \in \mathcal{S}_{x_i} \left(G_{x_i}(k,\ell) + \sum_{l=0}^{\ell-1} \boldsymbol{T_P}[l] \right), \forall x \in \{f, b, e\}. \tag{15}$$

B. Example of XMAP schedules

This section provides examples on how parameters $(\boldsymbol{T}_P, \boldsymbol{S}_\ell)$ can reproduce the schedule of XMAP architectures from the literature [10], [11] extended to N_P parallel decoder instances.

1) Classical XMAP schedule: one CD is computed in $T_{\text{CD}} = W_S + \lceil N_W/N_P \rceil - 1$ time slots before the next CD starts being processed. Consequently, $\boldsymbol{T_P}[\ell] = T_{\text{CD}}, \forall \ell$. Without overlaps, the choice of any particular window schedule does not influence the decoder convergence speed, since all precedence constraints of the interleaver are respected. Therefore, $\boldsymbol{S}_\ell$ can be simply defined in such a way that windows are processed in natural order: $\boldsymbol{S}_\ell[j] = \lfloor j/N_P \rfloor \ \forall j \in [0, N_W - 1], \forall \ell$. The resulting latency simplifies to $T_L = \ell_{\text{total}} \cdot T_{\text{CD}}$.

2) Shuffled processing schedule: Such decoders process all CDs in parallel and exchange (update) extrinsic information as soon they are generated. Therefore, we have

$$\boldsymbol{T_P}[\ell] = \begin{cases} 0, & \text{mod}_{N_{\text{CD}}} \ell > 0 \text{ or } \ell = 0, \\ T_{\text{CD}}, & \text{otherwise.} \end{cases} \tag{16}$$

The precedence constraint $\mathscr{C}_P$ is not verified as CDPs fully overlap. The resulting latency simplifies to $T_L = T_{\text{CD}} \lceil \ell_{\text{total}}/N_{\text{CD}} \rceil$. If $N_P < N_W$, the choice of the window schedule $\boldsymbol{S}_\ell$ may have an effect on the decoder convergence speed, i.e. the value of ℓ_{total} ensuring the error target is reached, and thus solving (8).

3) Iteration overlap schedule: The IOL technique [7] allows extrinsic information to be updated while the next CD is updating state metrics, i.e. *consuming* extrinsic information from the previous CD. The number of overlapping time slots between CDPs $(\ell-1, \ell)$ (overlap depth) $T_{\text{OD}}[\ell] = T_{\text{CD}} - T_P[\ell]$ is chosen such that the precedence constraints $\mathscr{C}_P(k)$ imposed by the interleavers in $\mathcal{C}$ are respected

$$\mathscr{C}_P(k): \ T_P[\ell] \geq \boldsymbol{S}_{\ell-1}\big[w_G(k)\big] - \boldsymbol{S}_\ell\big[w_C(k)\big] + \Delta_{UX}^{(\ell)}(k), \tag{17}$$

$\forall k$, with $w_C(k) = w_G\big(\Pi_i(\Pi_{i'}^{-1}(k))\big), i' = \text{mod}_{N_{\text{CD}}}(i-1)$. For the case where all XMAP windows are processed in parallel (UXMAP architecture), $\Delta_{UX}^{(\ell)}(k)$ is the delay between the time when the extrinsic information of bit index k is generated and the time when it is consumed between CDPs $\ell-1$ and ℓ. The choice of the window schedule should maximize T_{OD} while satisfying $\mathscr{C}_P$. The work in [7] proposed window schedule solutions that achieve a latency reduction of $20-25\%$ for most LTE interleaver configurations.

C. Advanced iteration overlap

Of the decoding schedules discussed in the previous section, the classical XMAP - and the IOL schedule - fully respect the precedence constraints $\mathscr{C}_P(k)$, while the shuffled processing schedule completely ignores them. In the following, we propose to extend the iteration overlap technique by allowing further overlap even when some precedence constraints are violated. We consider that the overlap depth $T_{\text{OD}}^\star$ is fixed for each CDP ($\boldsymbol{T}_{\text{OD}}[\ell] = T_{\text{OD}}^\star, \forall \ell$). This implies that $\boldsymbol{T_P}[\ell] = T_{\text{CD}} - T_{\text{OD}}^\star = T_P^\star, \ \forall \ell$. Furthermore, the overlap depth is limited to a maximum $T_{\text{OD,max}} = \min(T_{\text{CD}}/N_{\text{CD}}, W_L)$ which accounts for the available hardware resources since the underlying hardware model remains the same for the overlapped and non-overlapped cases. A dedicated optimization of the window schedule is not within the scope of this paper and will be subject of further study. The schedule is therefore fixed in the natural processing order: $\boldsymbol{S}_\ell[k] = k, \forall \ \ell$.

V. Evaluation

In this section, we evaluate the latency reduction achieved by the advanced IOL schedule proposed in Section IV-C for some 4G/LTE [1] turbo code configurations. All results are provided for the operating point $P_{e,\text{target}} = 10^{-3}$ of FER for a target signal-to-noise ratio target ζ, using a BPSK modulation and decoder update functions (f_f, f_b, f_e) based on the MLM algorithm. The latency T_L, expressed in number of time slots,

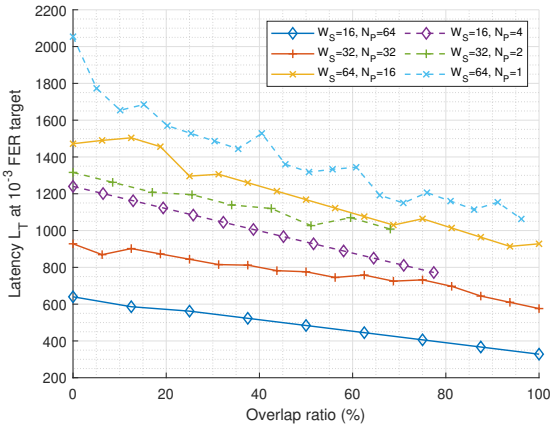

Figure 4: Latency as a function of the overlap ratio at a target FER of 10^{-3} for $K = 1024$, $C_R = 4/5$, $\zeta = 3.8$ dB.

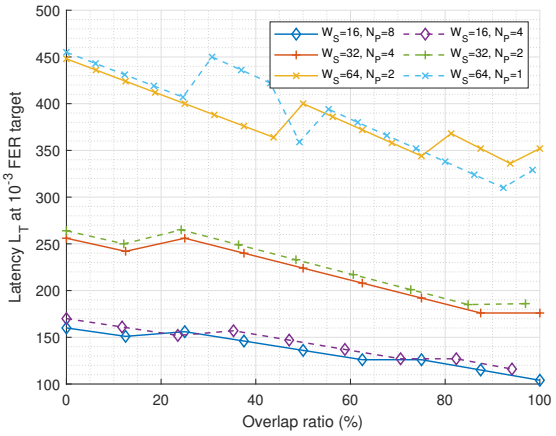

Figure 5: Latency as a function of the overlap ratio at a target FER of 10^{-3} for $K = 128$, $C_R = 0.323$, $\zeta = 2.8$ dB.

is evaluated by finding the required number of CDPs denoted by ℓ_{total} where the decoder achieves $P_{e,\text{target}}$. We show the results as a function of the overlap ratio R_{OD}, defined as the ratio between the chosen overlap depth and the maximum supported overlap depth: $R_{\text{OD}} = T^{\star}_{\text{OD}}/T_{\text{OD,max}}$.

A first example of the achieved latency is shown in Fig. 4 considering a code length $K = 1024$ bits, code rate of $C_R = 4/5$ and $\zeta = 3.8$ dB. Several configurations of window sizes and number of decoder instances (W_S, N_P) are evaluated. The results demonstrate that increasing the overlap ratio yields a significant reduction in decoding latency, whatever the choice of the window. Compared to a non-overlapped schedule ($R_{\text{OD}} = 0\%$), the achieved latency reduction ranges from 32% to 49% when fully overlapped ($R_{\text{OD}} = 100\%$). The overlap is particularly efficient for reducing latency in the case of $(W_S = 64, N_P = 1)$.

We evaluated the proposed schedule for a LTE turbo code configuration with $K = 128$ bits, $C_R = 0.323$ and $\zeta = 2.8$ dB. Fig. 5 shows that the reduced latency ranges between $20\%-30\%$ when $R_{\text{OD}} = 100\%$ which is still a significant reduction albeit lower than for $K = 1024$ bits.

VI. Conclusion

In this paper, we propose a novel framework for turbo decoding in which state metrics and extrinsic information can be updated in any order specified by a newly introduced decoding schedule parameter. This parameter is flexible enough to reproduce decoding schedules inferred from any hardware architecture and to minimize latency while meeting a target operating point defined by the error-rate and signal-to-noise ratio couple. Based on this, we propose an advanced iteration overlap XMAP schedule which achieves more than 30% latency reduction compared to typical XMAP schedules for the same FER target.

Acknowledgment

This work was partially funded by the ANR under the TurboLEAP project (ANR-20-CE25-0007) and under the France 2030 program, grant NF-PERSEUS (ANR-22-PEFT-0004).

References

[1] Third Generation Partnership Project, *LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (3GPP TS 36.212 version 17.1.0 Release 17)* , Apr. 2022.

[2] P. Schulz *et al.*, "Latency critical IoT applications in 5G: Perspective on the design of radio interface and network architecture," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 70–78, 2017.

[3] J. Zhang and M. P. C. Fossorier, "Shuffled iterative decoding," *IEEE Trans. on Commun.*, vol. 53, no. 2, pp. 209–213, Feb 2005.

[4] O. Muller, A. Baghdadi, and M. Jezequel, "Exploring parallel processing levels for convolutional turbo decoding," in *2nd Int. Conf. on Info. & Commun. Tech.*, vol. 2, 2006, pp. 2353–2358.

[5] S. Weithoffer, C. Abdel Nour, N. Wehn, C. Douillard, and C. Berrou, "25 Years of Turbo Codes: From Mb/s to beyond 100 Gb/s," in *Int. Symp. on Turbo codes and iter. proc. (ISTC)*, Dec 2018, pp. 1–6.

[6] S. Weithoffer, O. Griebel, R. Klaimi, C. Abdel Nour, and N. Wehn, "Advanced Hardware Architectures for Turbo Code Decoding Beyond 100 Gb/s," in *IEEE Wireless Commun. and Networking Conf. (WCNC 2020)*, Seoul, Korea (South), May 2019.

[7] S. Weithoffer, G. Aousaji, J. Nadal, and C. Abdel Nour, "Iteration Overlap for Low-Latency Turbo Decoding," in *Int. Symp. on Topics Coding (ISTC)*, 2023, pp. 1–5.

[8] Z. Yuping and K. K. Parhi, "High-Throughput Radix-4 logMAP Turbo Decoder Architecture," in *Proc. Fortieth Asilomar Conf. on Signals, Systems and Computers (ACSSC)*, Oct. 2006, pp. 1711–1715.

[9] R. G. Maunder, "A Fully-Parallel Turbo Decoding Algorithm," *IEEE Trans. on Commun.*, vol. 63, no. 8, pp. 2762–2775, Aug 2015.

[10] M. May, T. Ilnseher, N. Wehn, and W. Raab, "A 150Mbit/s 3GPP LTE Turbo code decoder," in *Design, Autom.and Test in Eu. Conf. (DATE)*, March 2010, pp. 1420–1425.

[11] S. Weithoffer, F. Pohl, and N. Wehn, "On the applicability of trellis compression to Turbo-Code decoder hardware architectures," in *Int. Symp. on Turbo Codes and iter. proc. (ISTC)*, Sep. 2016, pp. 61–65.

[12] N. Wiberg, H.-A. Loeliger, and R. Kotter, "Codes and iterative decoding on general graphs," in *Proc. of 1995 IEEE Int. Symp. on Inf. Theory*, 1995, p. 468.

[13] V. H. S. Le, C. Abdel Nour, E. Boutillon, and C. Douillard, "Revisiting the Max-Log-Map algorithm with SOVA update rules: new simplifications for high-radix SISO decoders," *IEEE Trans. Commun.*, vol. 68, no. 4, pp. 1991–2004, 2020.

[14] F. Zhai and I. Fair, "New error detection techniques and stopping criteria for turbo decoding," in *2000 Canadian Conf. on Electrical and Computer Engineering (CCECE)*, vol. 1, 2000, pp. 58–62 vol.1.

[15] S. Weithoffer and N. Wehn, "Latency reduction for lte/lte-a turbo-code decoders by on-the-fly calculation of crc," in *IEEE Int. Symp. Pers. Indoor Mobile Radio Commun. (PIMRC)*, 2015, pp. 1409–1414.

[16] N. Cassiau, L. Maret, J.-B. Doré, V. Savin, and D. Kténas, "Assessment of 5G NR physical layer for future satellite networks," in *IEEE Global Conf. on Sig. and Inf. Proc. (GlobalSIP)*, 2018, pp. 1020–1024.