Iteration Overlap for Low-Latency Turbo Decoding

Stefan Weithoffer, Ghazi Aousaji, Jérémy Nadal, Charbel Abdel Nour

Iteration Overlap for Low-Latency Turbo Decoding

Stefan Weithoffer, Ghazi Aousaji, Jeremy Nadal, Charbel Abdel Nour
IMT Atlantique, Department of Electronics, Lab-STICC - UMR 6285
Email: {stefan.weithoffer, rami.klaimi, charbel.abdelnour} @imt-atlantique.fr

***Abstract*—Achieving high decoding throughput and latency has been challenging for turbo decoders due to the limitations in terms of parallelism on component decoder level. To alleviate this issue, we propose an iteration overlap scheme able to apply a decoding schedule tailored to both, the decoder hardware architecture and the interleaver constraints. The proposal aims to minimize the achieved decoding latency without penalizing performance when compared to baseline decoders. To that end, we formulate the window schedule optimization problem when applying iteration overlap in pipelined Turbo Decoder hardware architectures. Then, we propose a method to find optimal window schedules under realistic assumptions. Results demonstrate that latency is reduced by $20-25\%$ for most Long Term Evolution (LTE) interleaver configurations. For specific interleavers, the achieved latency reduction can be as high as 62%. This indicates that further latency savings could be achieved if iteration overlap is taken into account when designing interleavers.**

***Keywords*—Forward Error Correction, Turbo decoder, Shuffled decoding, Low-latency.**

I. Introduction

Turbo codes [1] are a well-known code class that provides built-in rate flexibility with a low-complexity fast encoding. In the more than 30 years since their inception, they have found their way into serveral wireless communication standards such as LTE Advanced and DVB-RCS2 [2], [3]. Despite the contending Low Density Parity Check codes (LDPC) being chosen for the 3GPP 5G NR standard, Turbo codes will still be included in the continued evolution of LTE. Indeed recent advances in code design [4], decoding algorithm [5] and hardware architecture [6], [7] show that significant performance improvements are still being made.

An important performance metric for forward error correction for IoT is the decoding latency [8]. However, classical Turbo Decoder hardware architectures are limited in terms of parallelism on component decoder level and consequently in terms of achievable throughput and latency [6]. Shuffled decoding has been proposed as a way to extend parallel decoding to the iteration level by immediate extrinsic information exchange [9], [10]. However, it comes at a penalty in error correcting performance that has to be compensated for by additional decoding iterations. On another note, fully pipelining the decoding iterations allows very high decoding throughputs [6], [11]. There, a significant part of the implementation complexity is owed to first-in-first-out (FIFO) memories needed to buffer the extrinsic information between the half-iteration pipelines.

While for LDPC decoding, many works explored the problem of efficiently scheduling the decoding of different layers to mitigate the drawbacks of immediate (partial) information exchange [12], [13], to the best of our knowledge similar exhaustive works do not exist yet for Turbo Decoding.

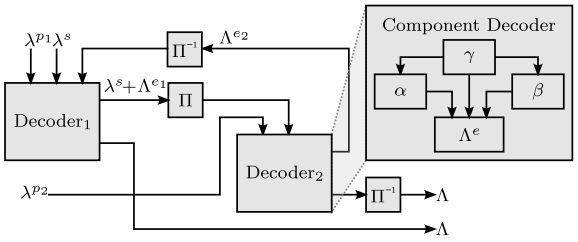

Figure 1: General Turbo Decoder structure.

In this work we formulate and solve the problem of efficiently scheduling the decoding of different windows while considering the underlying hardware architecture. The solution was applied to the case of pipelined Turbo Decoders. Latency evaluations for the codes of the LTE-A Pro standard [2] show an average latency saving of 20%-25% and individual savings for certain interleavers reaching as high as 62%.

The remainder of this paper is structured as follows: Section II briefly recalls background and context. In Section III, we discuss the proposed iteration overlap and formulate the scheduling problem. Section III-C presents a simplified solution to the problem for the case of the LTE standard. The corresponding latency savings are provided in Section IV before Section V concludes the paper.

II. Background

A turbo decoder consists of two component decoders connected by an interleaver Π and a de-interleaver Π^{-1} exchanging extrinsic information in an iterative loop (see Fig.1). We refer to an execution of one component decoder as a *Half Iteration* (HI). Most hardware architectures for turbo decoding implement one decoder instance based on the max-Log-MAP algorithm (MLM) which alternately operates as Decoder$_1$ and Decoder$_2$ [1]. The MLM computes the extrinsic information Λ^e from the recursively computed forward and backward state metrics α_k and β_k, and from the branch metrics of the code-trellis. In order to increase the throughput and lower the decoding latency, frames are split into smaller *sub-blocks* (or *"windows"* of size W_S) which are then processed using spatial and functional parallelism on component decoder level.

A. Parallel Turbo Decoder HW-Architectures

Parallel MAP (PMAP): The decoding of the windows is performed on P serial *sub-decoder cores* in parallel (spatial parallelism) [10], [14].

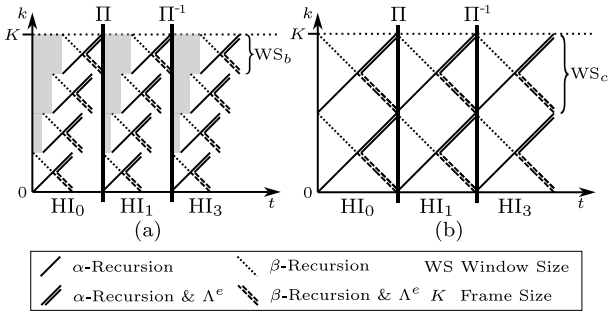

Figure 2: Windowing examples for (a) XMAP (b) UXMAP.

Fully Parallel MAP (FPMAP): The fully parallel MAP decoder proposed in [15] can be seen as the extreme case of the PMAP architecture with the P equal to the frame size and $W_S = 1$. It implements hardware instances for both component decoders which work in parallel and suffers from the same drawbacks as *shuffled decoding* [9] (see below).

Pipelined MAP (XMAP): The XMAP employs functional parallelism by unrolling the recursive α- and β-calculations onto a pipeline [16], [17]. Several windows of size $W_S = P$ are decoded in parallel while moving through the pipeline as illustrated in Fig. 2(a) for the decoding of a frame of size K split into 4 windows of size W_S over time t.

Fully Pipelined Iteration Unrolled MAP (UXMAP): This architecture extends the pipelining of windows to fully pipelining the iterative loop [6], [11]. Fig. 2(b) shows the resulting decoding schedule: Complete frames are processed in the UXMAP pipeline consisting of pipelined instances for each HI and each window allowing for a very high throughput.

While our results can also be applied to the PMAP, we will limit the following discussions regarding the proposed iteration overlap scheduling to its application in the context of XMAP and UXMAP for brevity and clarity.

B. Interleaving and Iteration Level Parallelism

In parallel decoder architectures, multiple Λ^e are generated at the same time (see again Fig. 2) and have to be interleaved in between HIs. For implementations with only one decoder instance, this requires parallel memory accesses which may lead to access conflicts. Therefore, modern interleavers like *Quadratic Permutation Polynomial* (QPP) [18] and *Almost Regular Permutation* (ARP) interleavers [19] are designed to be contention free for any P that divides the frame size K. However, extrinsic information exchange between successive decoding HIs $(\ell, \ell+1)$ must respect the precedence constraints imposed by Π and Π^{-1}. Indeed, meaningful extrinsic information at position i of the frame can only be exchanged and *consumed* (i.e. used for computation) by the next HI at time $C(i)$ once it has been generated by the previous HI at time $G(i)$. Therefore, the commonly used approach is to wait that all Λ^e are generated at HI ℓ before starting to consume them at HI $\ell+1$ which guarantees fulfilling the precedence constraints regardless of the interleaver. Fig. 3(a) illustrates this for the UXMAP architecture. However, memory has to be added in the form of FIFO (First-in-first-out) buffers to delay all but the last generated Λ^e, which increases circuit area, latency and energy consumption. For a given HI, the resulting latency L is largely determined as $L = \lceil W_S/2 \rceil$[1]($\lceil . \rceil$ denotes the ceil function). This latency accumulates at each processed HI.

Similarly, two (or more) XMAP decoder instances may be employed to process two consecutive HIs in parallel. In order to increase the throughput, the so called Shuffled decoding [9], [10] runs both decoder instances on the same frame and lets them exchange extrinsic information as soon as it is computed. In general, this leads to the component decoders not fully benefiting from the extrinsic information exchange. Indeed due to interleaving/de-interleaving, the extrinsic information needed by one component decoder may not yet have been computed by the other (i.e. precedence constraints of the form $k < \Pi(k)$ or $k > \Pi(k)$). Consequently, additional HI are needed in comparison to the non-shuffled case to compensate for the loss in error correction. This effect is well known to significantly lower the efficiency of shuffled decoding [10].

III. Iteration Overlap

We propose a method to define a schedule respecting all precedence constraints. While still maximizing the reduction in decoding latency compared to the baseline approach of waiting for the decoding of the HI to complete, the proposed schedule fully mitigates the need of additional HI incurred for classical shuffled decoding. We formulate and solve the respective scheduling problem to guarantee maximum *Iteration Overlap* (IOL) for given window size W_S and Π.

A. Iteration overlap

For the UXMAP with an IOL depth of OD, the extrinsic of bit index i, denoted by Λ_i^e and generated at time slot $G_{\text{UX}}(i)$ can start to be consumed by the next HI at time slot $C_{\text{UX}}(i)$ OD clock-cycles (time slots) earlier, resulting into a latency of $L_{\text{UX}} = L - \text{OD}$. This is illustrated in Fig. 3(b) where the grey computation units are shifted $\text{OD} = W_S/2 - 1 = 5$ cycles to become the green units, which represents a full overlap (one XMAP window is 12 cycles). Our aim is to determine the minimum UXMAP latency L_{UX} able to respect the precedence constraints (i.e. no performance penalty). It is directly related to the definition of the interleaver Π such that

$$G_{\text{UX}}(i) = \left\lfloor \left| \bmod(i, W_S) - \frac{W_S - 1}{2} \right| \right\rfloor, \tag{1}$$

$\lfloor . \rfloor$ denotes the floor function. Similarly, bit index i is consumed at time slot $C_{\text{UX}}(i) + L_{\text{UX}}$, with $C_{\text{UX}}(i) = \lfloor (W_S - 1)/2 \rfloor - G_{\text{UX}}(i)$. The precedence constraints impose that

$$\forall i \in [\![0, K-1]\!], C_{\text{UX}}\big(\Pi^{(\ell)}(i)\big) + L_{\text{UX}} \geq G_{\text{UX}}(i), \tag{2}$$

$$\implies L_{\text{UX}} = \max_{\forall i \in [\![0,K-1]\!]} \Big(G_{\text{UX}}(i) - C_{\text{UX}}\big(\Pi^{(\ell)}(i)\big) \Big), \tag{3}$$

where $\Pi^{(\ell)}$ is Π or Π^{-1} depending on HI ℓ being even/odd.

[1]Pipelined computation of Λ^e may incur an additional latency of several clock cycles that is omitted here for clarity [7].

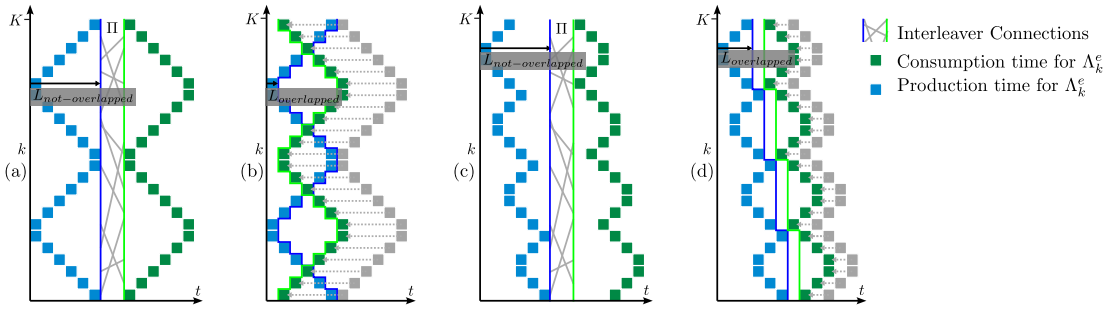

Figure 3: Illustration of the extrinsic generation and consumption between successive HIs ($(\ell, \ell+1)$) for a) UXMAP without overlap b) UXMAP with overlap c) XMAP without overlap d) XMAP with overlap.

In contrast to the UXMAP where all windows are processed in parallel, extrinsic information Λ^e is sequentially generated/consumed following the order of the processed windows in the pipeline for the XMAP. This is illustrated in Fig. 3(c) and the corresponding delay accumulates for each processed window. Without overlap, the resulting latency of a given HI is $L_X = L_{UX} + N_W$, with N_W denoting the number of windows within the frame of size K.

B. Problem Formulation

We define the *Scheduling of a window* as a length-N_W vector $\mathbf{S}_\ell$, with the k^{th} element denoting the time delay (corresponding to its position in the schedule) where window index k starts generating or consuming Λ^e. Note that each element of $\mathbf{S}_\ell$ must be distinct, i.e. $\mathbf{S}_\ell[k] \neq \mathbf{S}_\ell[k']$, $\forall k \neq k'$. In the example of Fig. 3(c), we have $\mathbf{S}_\ell = \mathbf{S}_{\ell+1} = [3, 2, 1, 0]$. At bit index i, the generation and consumption time slots are

$$G(i) = G_{\text{UX}}(i) + \mathbf{S}_\ell\big[w_G(i)\big], \tag{4}$$

$$C(i) = C_{\text{UX}}(i) + L_{\text{X}}^{(\ell)} + \mathbf{S}_{\ell+1}\big[w_C(i)\big], \tag{5}$$

where $w_G(i) = \lfloor i/W_S \rfloor$ and $w_C(i) = w_G\big(\Pi^{(\ell)}(i)\big)$ are the generation and consumption window indexes to which bit i belongs, and $L_{\text{X}}^{(\ell)}$ corresponds to the latency at HI ℓ. The precedence constraints impose[2]

$$L_X^{(\ell)} = \max_{\forall i} \Big(\mathbf{S}_\ell\big[w_G(i)\big] - \mathbf{S}_{\ell+1}\big[w_C(i)\big] + \Delta_{UX}^{(\ell)}(i) \Big), \tag{6}$$

where $\Delta_{UX}^{(\ell)}(i) \triangleq G_{\text{UX}}(i) + C_{\text{UX}}\big(\Pi^{(\ell)}(i)\big)$ corresponds to the difference in delay between the time when the extrinsic information of bit index i is generated and the time when it is consumed between HI ℓ and $\ell+1$.

Constraints of (6) can be relaxed since only the connection corresponding to the largest delay $\Delta_{UX}^{(\ell)}(i)$ sets the timing constraint for all Λ^e sharing the same generation window n and consumption window m, independently of the schedule. Defining the set $\mathcal{E}_{n,m} = \{i : \ w_G(i) = n, w_C(i) = m\}$ as the list of all bit indexes i that belong to generation window $n = w_G(i)$ and consumption window $m = w_C(i)$ we have $\forall (n,m) \in [\![0, N_W - 1]\!]^2$,

$$\Delta_{n,m}^{(\ell)}(\mathbf{S}_\ell, \mathbf{S}_{\ell+1}) \triangleq \mathbf{S}_\ell[n] - \mathbf{S}_{\ell+1}[m] + D_{n,m}^{(\ell)}. \tag{7}$$

Here, $D_{n,m}^{(\ell)}$ is the local delay between generation window n and consumption window m. It is expressed[3] as

$$D_{n,m}^{(\ell)} = \max_{\forall i \in \mathcal{E}_{n,m}} \Big(G_{\text{UX}}(i) - C_{\text{UX}}\big(\Pi^{(\ell)}(i)\big) \Big). \tag{8}$$

Finally, (6) can be rewritten as

$$L_X^{(\ell)} = \max_{\forall (n,m)} \Delta_{n,m}^{(\ell)}(\mathbf{S}_\ell, \mathbf{S}_{\ell+1}). \tag{9}$$

Ideally, the window schedules must be chosen to minimize the accumulated latency up to the last iteration, i.e. after $\ell_{\max}$ HIs. This is equivalent to solving the following problem

$$\mathscr{P}_0: \quad \text{minimize} \quad \sum_{\ell=0}^{\ell_{\max}-1} \max_{\forall (n,m)} \Big(\Delta_{n,m}^{(\ell)}(\mathbf{S}_\ell, \mathbf{S}_{\ell+1}) \Big), \tag{10}$$

$$\text{s.t. } \mathscr{C}_0: \quad \forall \ell, \forall (k,k') \setminus k \neq k', \mathbf{S}_\ell[k] \neq \mathbf{S}_\ell[k']. \tag{11}$$

It can be shown that problem $\mathscr{P}_0$ is an integer linear programming problem (ILP), and is computationally expensive to solve for a large number of windows. In the next sub-section, we propose to simplify $\mathscr{P}_0$, adapting it to more practical usages, and we provide a simple way to find the optimal schedule.

C. Problem simplification and solution

In practice, iterative turbo decoders employ iteration control [20]. Therefore, it is difficult to predict the target number of HIs for solving $\mathscr{P}_0$. Instead, it is more practical to optimize the scheduling of windows on a per-HI basis: The obtained optimal schedule vector at the previous HI $\ell-1$, denoted $\mathbf{S}_{\ell-1}^\star$, is used as generation window schedule when optimizing the schedule vector at the next HI ℓ. This iterative optimization procedure can be expressed as

$$\mathbf{S}_\ell^\star = \underset{\mathbf{S}_\ell}{\text{argmin}} \ \max_{\forall m} \Big(\Delta_m^{(\ell)}(\mathbf{S}_{\ell-\mathbf{1}}^\star) - \mathbf{S}_\ell[m] \Big), \tag{12}$$

with $\Delta_m^{(\ell)}(\mathbf{S}_{\ell-\mathbf{1}}^\star) = \max_n \big(D_{n,m} + \mathbf{S}_{\ell-\mathbf{1}}^\star[n] \big)$. In addition, received information generally enters the decoder in natural

[2] In practice, achievable latency is bounded by the processors capability to consume Λ^e. This hardware constraint is omitted here.

[3] To avoid mathematical inconsistency if $|\mathcal{E}_{n,m}| = 0$, we extend the definition the operator max as follows: $\forall x, \ \max_\emptyset x = -\infty$

order, and thus without loss of generality, we select the first window schedule to be $\mathbf{S}_0^\star[k] = k$ for all k windows. Now, (12) corresponds to the solution of the following ILP

$$\mathscr{P}_1: \quad \text{minimize } L_X^{(\ell)}, \tag{13}$$

$$\text{s.t. } \mathscr{C}_0, \mathscr{C}_1: \forall m, \mathbf{S}_\ell[m] \geq \Delta_m^{(\ell)}(\mathbf{S}_{\ell-\mathbf{1}}^\star) - L_X^{(\ell)}. \tag{14}$$

In constraint $\mathscr{C}_1$, $\Delta_m^{(\ell)}$ can be sorted by ascending order, from the less constraining delay to the most constraining delay. The notation $\overrightarrow{m} = \text{argsort}(\Delta_m^{(\ell)})$ indicates the sorted index. If $\mathscr{C}_1$ is not fulfilled when using the schedule vector $\mathbf{S}_\ell[\overrightarrow{m}] = m$, $m \in [\![0, N_W - 1]\!]$, for a given latency value $L_X^{(\ell)}$, then it is not possible to find another schedule vector that would satisfy $\mathscr{C}_1$. Indeed, if $m < \Delta_{\overrightarrow{m}}^{(\ell)}(\mathbf{S}_{\ell-\mathbf{1}}^\star) - L_X^{(\ell)}$, then it is not possible to find any $m' > m$ to exchange m with, since $\Delta_{\overrightarrow{m'}}^{(\ell)} \geq \Delta_{\overrightarrow{m}}^{(\ell)}$. Then, the minimum latency value is determined by the most constraining extrinsic bit index to schedule:

$$L_X^{\star(\ell)} = \max_{m \in [\![0, N_W-1]\!]} \left(\Delta_{\overrightarrow{m}}^{(\ell)} - m \right). \tag{15}$$

Note that both problems $\mathscr{P}_0$ and $\mathscr{P}_1$ can be easily solved if 1) Π is designed to achieve full overlap and 2) all generation windows are connected to all consumption windows ($|\mathcal{E}_{n,m}| = N_W^2$). There, we have $D_{n,m} = 0\ \forall(n,m)$, and $\max_{n,m}(\mathbf{S}_\ell[n] - \mathbf{S}_{\ell+1}[m]) = N_W - 1$ and thus, the latency would always be $\ell_{\max}(N_W - 1)$, independent of schedule.

D. Adaptability to different architecture parameters

In order to find an optimal schedule when considering variable architectural choices, the problem formulation needs to be adapted. We give the following examples: First, for XMAP decoders that implement 2^r-radix computation units [7], [14], bit index i is consumed at $C_{\text{UX}}(i,r) = \lfloor C_{\text{UX}}(i)/r \rfloor$ and generated at $G_{\text{UX}}(i,r) = \lfloor G_{\text{UX}}(i)/r \rfloor$. In problem $\mathscr{P}_1$, this only affects the computation of $\Delta_m^{(\ell)}$, and the proposed method to find the optimal schedule is the same as the one proposed in Section III-C. Furthermore, it is possible to instantiate Q XMAP processors in parallel to speed up the extrinsic computation. This implies that Q times more extrinsic information can be pipelined at each clock cycle, with a total of $N_Q = \lceil N_W/Q \rceil$ windows computed per processor and per HI. The achievable latency can be reduced to a minimum of N_Q clock-cycles. Each processor can be scheduled differently, with $\mathbf{S}_{\ell,q}^\star \in [\![0, N_Q - 1]\!]^{N_Q}$ being the window schedule vector associated to processor q. These schedules can be derived from the optimal windows schedule $\mathbf{S}_\ell^\star$ obtained by solving $\mathscr{P}_1$, in such a way that concatenating each $\mathbf{S}_{\ell,q}^\star$ reproduces $\mathbf{S}_\ell^\star$. The corresponding concatenation procedure can be described through the following equation

$$\mathbf{S}_{\ell,p}^\star[m] = \left\lfloor \frac{1}{Q} \mathbf{S}_\ell^\star[m + Qp] \right\rfloor. \tag{16}$$

IV. Latency Evaluation

We evaluate the achievable latency reduction through the proposed IOL for several hardware architectures in comparison to the baseline scheme (without overlap). We consider the QPP interleavers defined in the LTE standard for the 188 frame lengths, from $K = 40$ to $K = 6144$ bits [2].

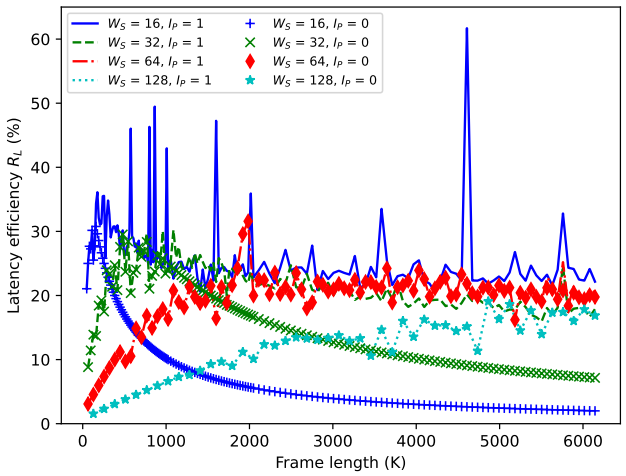

Figure 4: IOL latency efficiency for the 188 LTE frame lengths and $W_S \in \{16, 32, 64, 128\}$ (radix-2 MAP).

The total latency L_{DEC} for decoding up to $\ell_{\max}$ HIs is the contribution of two latency terms, L_{PROC} and L_{EXCH}. The first term $L_{\text{PROC}} = \ell_{\max} \lfloor W_S/(2r) \rfloor + L_{\text{I/O+ACQ}}$ represents the processing delay of XMAP processors that includes the delays incurred by the input/output interface and initial acquisitions denoted by $L_{\text{I/O+ACQ}}$ [17]. In general, the latency $L_{\text{I/O}}$ is comparatively small and the acquisition may be replaced by next-iteration initialization (NII) [21]. Therefore we neglect its effect for simplicity ($L_{\text{I/O+ACQ}} = 0$). The second term L_{EXCH} corresponds to the accumulated delay when extrinsic information is exchanged between successive HIs. For both baseline and IOL cases, it can be expressed as

$$L_{\text{EXCH}}[\text{baseline}] = \ell_{\max} \left(\left\lceil \frac{W_S}{2r} \right\rceil + N_Q \right), \tag{17}$$

$$L_{\text{EXCH}}[\text{IOL}] = \sum_{\ell=0}^{\ell_{\max}-1} \max\left(L_X^{\star(\ell)}, (1 - I_P) N_Q \right), \tag{18}$$

where $I_P = 1$ for architectures that allow iteration level parallelism such as UXMAP, otherwise $I_P = 0$. We evaluate the latency efficiency R_L defined as the latency reduction achieved by IOL when compared to the baseline case by

$$R_L = 1 - \frac{L_{\text{EXCH}}[\text{IOL}] + L_{\text{PROC}}}{L_{\text{EXCH}}[\text{baseline}] + L_{\text{PROC}}}. \tag{19}$$

Fig. 4 shows the obtained R_L values for the 188 LTE frame lengths and 4 different window sizes $W_S \in \{16, 32, 64, 128\}$ when considering one radix-2 XMAP instance with and without HI parallelism. The number of HIs is fixed to $\ell_{\max} = 16$. $I_P = 1$ implies that IOL can be fully exploited to improve R_L. The achieved values of R_L tend to converge to around $20\% - 25\%$ for the largest frame sizes ($K > 2000$) and most window sizes. It can be observed that R_L variance is significantly larger for small W_S values, with a peak of

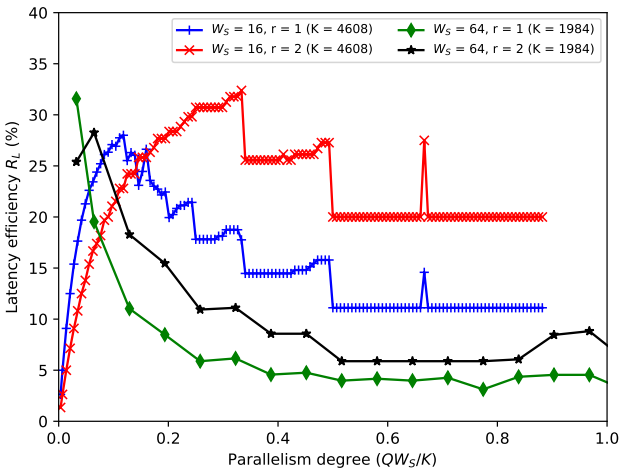

Figure 5: R_L vs. no. of XMAP Q ($K = 4608, W_S = 16$).

$R_L = 62\%$ at $K = 4608$ bits. Indeed, a smaller W_S increases the connection sparsity between windows, allowing more degrees of freedom during schedule optimization.

Without HI parallelism ($I_P = 0$), the latency efficiency is noticeably limited for the largest frame sizes when the smallest W_S are used. This problem can be alleviated by increasing the number of XMAP processors Q and the radix order r. The corresponding effect was evaluated in Fig. 5. This latter plots R_L for several Q and r values and for two of the best latency efficiency achieving sets of parameters (interleaver lengths $K = 4864$ and $K = 1984$ bits with $W_S = 16$ and $W_S = 64$) in Fig. 4. The x-axis of Figure 5 is the parallelism degree defined as the number of processed bits in parallel and normalized by the frame length (QW_S/K). For both frame lengths, the latency efficiency is further improved when using radix-4 MAP ($r = 2$), particularly for small window sizes. Increasing the number of processors improves R_L up to an optimal point $Q^\star$. Passed this threshold, R_L decreases since the parallelism level tends to the full parallel case where the precedence constraints of the interleaver supersede the schedule choice to become the only limiting factor for minimizing the latency. For the cases where $W_S = 16$, the maximum latency efficiency is obtained at $Q^\star = 34$ ($R_L = 28\%$) for radix-2 and $Q^\star = 96$ ($R_L = 32.38\%$) for radix-4.

V. Conclusion

We proposed in this paper an iteration overlap technique to reduce the latency of turbo decoders that fully mitigates the drawbacks of classical shuffled decoding. We formalized the window schedule optimization problem for minimizing latency as well as a method to find the best window schedule when considering practical simplifications. Results show that large latency reductions are obtained for several LTE interleavers without added hardware complexity. Furthermore, it is expected that latency can be further reduced if the interleaver is carefully designed to exploit the proposed iteration overlap technique.

Acknowledgment

This work was partially funded by the French National Research Agency TurboLEAP project (ANR-20-CE25-0007).

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes," in *IEEE Int. Conf. on Commun. (ICC)*, vol. 2, May 1993, pp. 1064–1070 vol.2.

[2] 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.

[3] ETSI, *Digital Video Broadcasting (DVB); Second Generation DVB Interactive Satellite System (DVB-RCS2); Part 2: Lower Layers for Satellite standard (ETSI EN 301 545-2 V1.3.1 (2020-07))*, Apr. 2020.

[4] R. Garzón-Bohórquez, C. Abdel Nour, and C. Douillard, "Protograph-based interleavers for punctured turbo codes," *IEEE Trans. on Commun.*, vol. 66, no. 5, pp. 1833–1844, May 2018.

[5] V. Le, C. A. 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.

[6] 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.

[7] S. Weithoffer, R. Klaimi, C. Abdel Nour, N. Wehn, and C. Douillard, "Low-complexity computational units for the local-SOVA decoding algorithm," in *IEEE 31st Int. Symp. Personal, Indoor and Mobile Radio Commun. (PIMRC)*, London, UK, Sept. 2020.

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

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

[10] 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.

[11] S. W. O. Griebel, R. Klaimi, C. A. 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.

[12] P. Radosavljevic, A. de Baynast, and J. Cavallaro, "Optimized Message Passing Schedules for LDPC Decoding," in *39th Asilomar Conf. on Signals, Systems and Comp.*, 2005.

[13] V. P. L., M. M. Marković, D. M. E. Mezeni, L. V. Saranovac, and A. Radošević, "Flexible High Throughput QC-LDPC Decoder With Perfect Pipeline Conflicts Resolution and Efficient Hardware Utilization," *IEEE Trans. on Circ. and Syst. I*, vol. 67, no. 12, pp. 5454–5467, 2020.

[14] Z. Yuping and K. K. Parhi, "High-Throughput Radix-4 logMAP Turbo Decoder Architecture," in *Proc. Fortieth Asilomar Conference on Signals, Systems and Computers ACSSC '06*, Oct. 2006, pp. 1711–1715.

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

[16] 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.

[17] 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.

[18] J. Sun and O. Y. Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," *IEEE Trans. on Inf. Theory*, vol. 51, no. 1, pp. 101–119, Jan. 2005.

[19] C. Berrou, Y. Saouter, C. Douillard, S. Kerouedan, and M. Jezequel, "Designing good permutations for turbo codes: towards a single model," in *IEEE Int. Conf. on Commun. (ICC)*, June 2004, pp. 341–345.

[20] F. Gilbert, F. Kienle, and N. Wehn, "Low complexity stopping criteria for UMTS turbo-decoders," in *57th IEEE Vehi. Tech. Conf. VTC Spring*, vol. 4, 2003, pp. 2376–2380.

[21] J. Dielissen and J. Huiskens, "State Vector Reduction for Initialization of Sliding Windows MAP," in *Int. Symp. on Turbo codes and iter. proc. (ISTC)*, Brest, France, Sep. 2000, pp. 387–390.