Towards an optimal parallel decoding of turbo codes

David Gnaedig *, Emmanuel Boutillon+, Jacky Tousch *, Michel Jézéquel –

* TurboConcept, 115 rue Claude Chappe, 29280 PLOUZANE, France
+ LESTER – Unité CNRS FRE 2734, UBS, BP 92116, 56321 LORIENT Cedex, France
– GET/ENST Bretagne/PRACOM – Unité CNRS UMR 2658, CS 83818, 29238 Brest Cedex 3, France

E-mail: david.gnaedig@turboconcept.com, emmanuel.boutillon@univ-ubs.fr,
jacky.tousch@turboconcept.com, michel.jezequel@enst-bretagne.fr

Abstract

High throughput decoding of turbo-codes can be achieved thanks to parallel decoding. However, for finite block sizes, the initialisation duration of each half-iteration reduces the activity of the processing units, especially for higher degrees of parallelism. To solve this issue, a new decoding scheduling is proposed, with a partial processing overlapping of two successive half iterations. Potential memory conflicts introduced by this new scheduling are solved by a constrained interleaver design. An example of application of the proposed technique shows that the complexity of the decoder is reduced by 25 % compared to a conventional approach.

1. Introduction

Since the introduction of turbo codes, there has been a large demand for high data rate, low complexity, turbo decoders. To avoid the duplication of memories required by the iterative decoding process, parallel decoding structures working on a single frame have been proposed. The frame is divided into sub-blocks that are simultaneously and independently decoded by several Processing Units (PUs) associated with as many memory banks (MBs) and performing a sliding window (SW) algorithm [1]. Such a parallel architecture leads to memory conflicts called parallel conflicts which occur when two PUs try to access the same MB. They are caused by the presence of the interleaver in the coding scheme. This issue has been widely addressed in the literature and the proposed solutions can be classified into three types. They are characterized, by analogy with software development terminology, by the stages at which they tackle the issue of concurrent accesses (ranked from the lowest level to the highest level):

- "execution" stage solutions: during the execution of the parallel processing, the concurrent accesses are handled by additional hardware specially designed to solve the parallel conflicts [2].
- "compilation" stage solutions: during the design of the parallel decoding architecture a memory mapping preventing the emergence of any parallel conflict is found [3].
- "design" stage solutions: the interleaver and the architecture are designed jointly in order to guarantee the absence of parallel conflicts [4].

These solutions have succeeded in finding architectures enabling an efficient usage of memory resources. However, another issue arises when dealing with parallel architectures: the efficient usage of the computational resources. The idea behind this optimization is that it is not efficient to increase the number of PUs if the computational power they offer is used only partially. This issue is particularly critical in the context of sequential decoding of turbo codes where half-iterations are processed sequentially. Due to data dependencies and pipeline stages in the hardware implementation, the SW algorithm leads to idle time at the beginning and ending of each sub-block processing. This idle time can waste a significant part of the total processing power when the sub-block size is short, *i.e.*, short codeword length and/or high degree of parallelism. This paper addresses the efficiency of parallel turbo decoding architectures and gives efficient solutions for increasing significantly the activity of the PUs. To the best of our knowledge, this issue has not been considered so far in the literature.

This paper is divided into six sections. In section 2, a model of the throughput and activity of the turbo decoder is proposed. This model is applied in section 3 to evaluate the efficiency of two different SW decoding schedules in the context of parallel decoding. Section 4 proposes a solution to increase the activity of the PUs in the context of iterative decoding. The idea is based on a processing overlapping of consecutive half-iterations. The resulting conflicts are handled at the design stage by a new joint interleaver-architecture methodology proposed in section 5. Section 6 concludes and summarizes the performance results and the complexity reductions.

2. Throughput and activity of the turbo decoder

This section introduces the issue of the implementation of high-throughput turbo decoder. In particular, we will introduce an analytical model for the expression of the throughput of a turbo decoder architecture. We will derive a measure of its efficiency by evaluating the ratio between the throughput and the complexity.

2.1 Model of the throughput

Let us first introduce a simplified model of a turbo decoder that takes into account the architectural constraints. To this end, we make the two following assumptions. On the algorithmic side, the computational complexity of the Max-Log-MAP algorithm is evaluated as the total number of additions and comparisons required to perform one half decoding iteration (in fact, these two operations are of equivalent complexity). On the architecture side, we assume that the decoder is composed of Γ physical elementary operators (one operator performs indifferently additions or comparisons). Under these assumptions, the decoded throughput (in bits/sec) for a two dimensional m-binary turbo code can be expressed as

$$D_b = m \cdot \alpha \cdot \frac{f_{clk} \cdot \Gamma}{2 \cdot I_{\max} \cdot \chi_u}, \qquad (1)$$

where:

- f_{clk} is the clock frequency of the architecture (a fully synchronous design is assumed).
- Γ is the number of elementary operators of architecture. They enable a maximal computational power of Γ elementary operations per clock cycle to be executed. This parameter is proportional to the computational complexity of the architecture. The product $f_{clk} \cdot \Gamma$ corresponds to the available computational power per second.
- I_{max} is the number of turbo decoding iterations.
- χ_u represents the computational complexity of one trellis stage of the Max-Log-MAP algorithm in number of elementary operations. The product $2 \cdot I_{max} \cdot \chi_u$ corresponds to the computational complexity of the algorithm.
- α is called the activity of the Pus and corresponds to the average utilization of the computational resources during the execution of the algorithm.

$$\alpha = \frac{\chi_u \cdot n_T}{\Gamma \cdot T}. \qquad (2)$$

where n_T correspond to the number of trellis stages processed during a period of T cycles.

The concept of activity is very important since it measures the efficiency of the architecture. In the ideal case $\alpha = 1$, *i.e.* the architecture is efficient and all computational resources are always used. Otherwise, when $\alpha < 1$ there are some resources that are used only partially resulting in a lower architecture efficiency. Obviously, α depends on the algorithm, especially the data dependencies induced by the algorithm and also on the resources that are implemented.

2.2 Validation of the model

Our study uses the Max-Log-MAP algorithm, which is composed of three main computational blocks per trellis stage: a computation of the forward state metrics (SMs), a computation of the backward SMs and a computation of the soft outputs (SOs) obtained by combining the forward and backward SMs.

We have evaluated the number of elementary operations needed by these three main blocks for a double-binary 8-state RSC code of memory $\nu = 3$, that uses $m = 2$ input bits to produce $n = 4$ output bits. The computational complexities relative to the computation of the SMs (either forward or backward) and the SOs are $\chi_{SM} = 72$ and $\chi_{SO} = 74$ elementary operations, respectively. Using the Max-Log-MAP algorithm, the total number of elementary operations χ_u needed for the production of the soft outputs for one trellis stage is thus given by

$$\chi_u = 2\chi_{SM} + \chi_{SO} = 218. \qquad (3)$$

Therefore, the SM and SO computational blocks represent a fraction among the overall complexity of 33 % and 34 %, respectively.

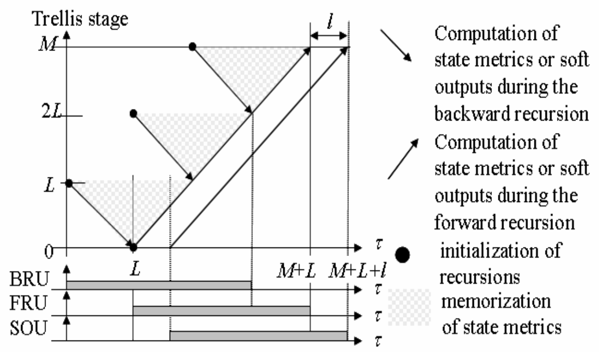
A more accurate evaluation of the complexity of the computational units has been obtained by the synthesis of a functional turbo decoder in a 0.18µ technology. The total complexity of the processing units amounts to 15398 gates and the SM and SO blocks represent a fraction among the overall complexity of 31 % and 38 %, respectively. These synthesis results demonstrate the validity of our model for evaluating the complexity of the algorithm. In particular, the relative complexities for each module are equivalent for the circuits considered. Therefore, in the following of the manuscript, the complexity of the Max-Log-MAP algorithm and the activity figures will be obtained by counting the number of elementary operators.

3. Decoding schedules

Different SW algorithms are described in the literature, differing mainly in the schedule used for the respective forward and backward processings, and by the SM initialisation scheme. We consider here that the same schedule is applied simultaneously to the P sub-blocks of size M trellis stage. The size of the frame in number of m-binary symbols is denoted N (N is also the number of trellis stages of the whole trellis). The initial values of the SMs are the corresponding metrics computed in the previous iteration, as described in [5]. After a brief review of two main schedules and the calculation of their activities, the last part of this section will compare their activities and will bring to the forth the decrease in the architecture efficiency with the increase in the degree of parallelism.

3.1 Schedule SW-Σ $^-$

This schedule is depicted in Fig. 1, where we adopt the graphical representation proposed in [6]. The horizontal axis represents the time τ, with units of clock periods (it is assumed that one complete stage of forward or recursion can be performed in exactly one clock period). The vertical axis represents the indices of the stages of the trellis. It is assumed that the soft inputs (observation data and a priori information) related to the M symbols are stored in a memory and are all available at time $\tau = 0$. For the sake of clarity of the figure, the pipeline latency of l cycles corresponding to the pipeline stages of a real architecture is represented at the end of the data path, *i.e.* at the computation of the soft outputs. However, in the general case, the pipeline registers are uniformly distributed along the data path.



Trellis stage
M
2L
L
0
l
L
M+L
M+L+l
τ
BRU
FRU
SOU
Computation of state metrics or soft outputs during the backward recursion
Computation of state metrics or soft outputs during the forward recursion
initialization of recursions
memorization of state metrics

Fig. 1 Execution and resource allocation of schedule *SW-Σ*$^-$.

The sub-block is divided into non-overlapping windows of L trellis stages on which a SW algorithm is performed. The backward recursion precedes the forward recursion and the backward state metrics are stored in memory BSM MEM. They are read out from BSM MEM during the forward recursion to be combined to the forward state metrics to produce the corresponding soft outputs.

The architecture for implementing this schedule requires the following computational units:

- a backward recursion unit (BRU) computing backward state metrics that are stored into BSM MEM.
- a forward recursion unit (FRU) computing the forward state metrics.
- a soft output unit (SOU) computing the soft outputs using the forward state metrics and the backward state metrics read out from the state metric memory BSM MEM.

Therefore, the complexity of an architecture performing the schedule *SW-Σ*$^-$ is equal to

$$\Gamma_{SW-\Sigma^-} = 2\cdot\chi_{SM} + \chi_{SO}. \tag{4}$$

The shaded areas in the lower part of Fig. 1 represents the time when the three computation units FRU, BRU and SOU are working, *i.e.* processing data. Since, with schedule *SW-Σ*$^-$, each of the $\Gamma_{SW-\Sigma^-}$ operators is used M times during M+L+l cycles, putting (3) and (4) into (2), yields the activity of *SW-Σ*$^-$:

$$\alpha_{SW-\Sigma^-} = \frac{M}{M+L+l} = \frac{N/P}{N/P+L+l}. \tag{5}$$

Due to the term L in the denominator, the activity decreases for small frame sizes. On the contrary, when the size of the frame tends toward infinity, the activity tends toward 1.

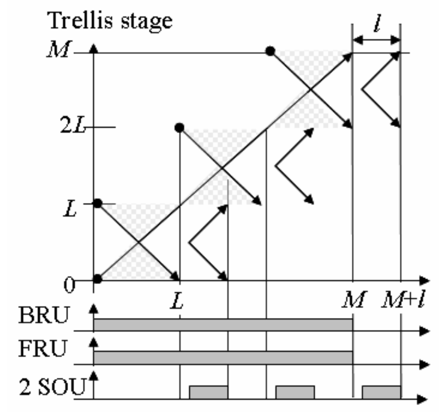


Fig. 2 Execution and resource allocation of schedule *SW-Σ*^{0}.

3.2 Schedule SW-Σ ^{0}

In schedule *SW-Σ*^{0} proposed in [7] and shown in Fig. 2, the forward and backward recursions are computed concurrently. From time $\tau = 0$ to time $\tau = L/2$, the forward and backward recursion units process the symbols 0 to $L/2 - 1$ and the symbols L-1 down to $L/2$, respectively, and store the corresponding forward and backward state metrics. The two recursions cut across each other at time $\tau = L/2$. Then, from time $\tau = L/2$ to L, the soft outputs are produced along with the two recursions: the forward (backward) state metrics produced by the forward recursion (respectively backward) are combined to the previously stored backward (respectively forward) state metrics. Note that *SW-Σ*^{0} has the same algorithmic computational complexity and memory requirements as *SW-Σ*$^-$. The half-iteration decoding duration is reduced to M+l cycles, however. This reduced decoding duration comes at the expense of a more complex implementation, since two soft outputs for two trellis stages are computed concurrently, thus requiring two SOU. The complexity of this schedule is given by:

$$\Gamma_{SW-\Sigma^0} = 2\cdot\chi_{SM} + 2\cdot\chi_{SO} \tag{6}$$

It is worth noting that the SOUs are used less than half of the time, which has a significant impact on the activity. Let $\beta_{SW-\Sigma^0}$ represent the ratio of computational complexity χ_u over the available computational complexity of the architecture $\Gamma_{SW-\Sigma^0}$, $\beta_{SW-\Sigma^0} = \chi_u / \Gamma_{SW-\Sigma^0}$. Using the expressions of χ_{SM} and χ_{SO},

$\beta_{SW-\Sigma^0} \approx 0.74$. Putting (3) and (6) into (2) and introducing $\beta_{SW-\Sigma^0}$ yields the expression of the activity

$$\alpha_{SW-\Sigma^0} = \beta_{SW-\Sigma^0} \frac{N/P}{N/P+l}. \quad (7)$$

The second term of this product shows that the activity of *SW-Σ*^{0} is independent of the window length *L*. Thus the throughput loss for small frames is reduced compared to *SW-Σ*$^-$. Unfortunately, due to the first term $\beta_{SW-\Sigma^0}$, the architecture is rather inefficient, even for long frames: when the size of the sub-block tends towards infinity, the activity tends toward this first term, while it tended toward 1 for schedule *SW-Σ*$^-$.

3.3 Activity comparison

The respective activities of schedules *SW-Σ*$^-$ and *SW-Σ*^{0} are compared in Fig. 3. For a single PU implementation (P = 1, M = N), schedule *SW-Σ*$^-$ has a higher activity than schedule *SW-Σ*^{0} for sub-block sizes above 70 trellis stages (see Fig. 3. case a). On the contrary, when sub-block size is reduced, (due to higher parallelism or small frames processing) the two activities drop, and the schedule *SW-Σ*^{0} becomes more efficient. The activity drop can be dramatic. For example, with schedule *SW-Σ*^{0}, about half of the computational resources are not used if P = 8 PUs decode a frame of size N = 160 trellis stages (see Fig. 3. case b). This explains the need for other techniques for increasing the activities of parallel turbo decoder implementations.

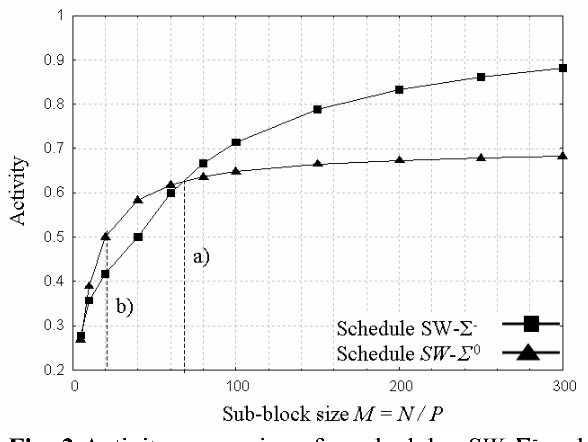


Fig. 3 Activity comparison for schedules *SW-Σ*$^-$ and *SW-Σ*^{0} as a function of the size of the trellis section $M = N/P$.

4. Increasing the activity

There are several directions that can be followed in order to increase the activity of the processing units. One can change the schedule of a SW algorithm, but whatever the schedule, this solution still suffers from the constant value l due to the pipeline registers. Another direction would be to process simultaneously the two dimensions of the turbo code as proposed in [8]. The solution that is proposed in this paper maintains the sequentiality of the half-iterations and improves the activity of the schedule *SW-Σ*$^-$.

4.1 Overlapping of half-iterations

In the context of iterative decoding of turbo codes, a significant activity improvement, virtually reaching the optimal value 1 could be obtained by starting the processing of the next half-iteration before the end of the current half-iteration as shown in Fig. 4. The depth of the overlapping region expressed in number of clock cycles is denoted Δ. This new schedule is denoted *OSW-Σ*$^-$(*Δ*) (also denoted *OSW-Σ*$^-$ for simplicity). The overlapping depth *Δ* is bounded by $\Delta_{max} = L + l$ which is the idle time of each of to the computation resources of schedule *SW*-Σ$^-$. When $\Delta = \Delta_{max}$ the activity of SW-Σ$^-$ is raised to 1 (no idle time left).

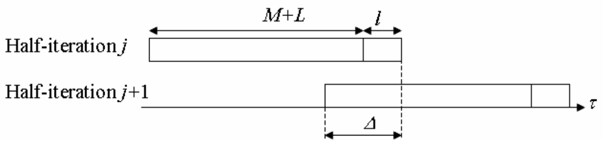


Fig. 4 Overlapping of two consecutive half-iterations with an overlapping depth *Δ*.

Unfortunately, this solution leads to another type of memory conflict, which we term a consistency conflict: due to interleaving/de-interleaving the extrinsic inputs required at the beginning of the next half-iteration might not yet have been produced by the current half-iteration. The next section describes this phenomenon in more details.

4.2 Consistency conflicts

In order to illustrate a consistency conflicts, let us use a two-dimensional representation of an interleaver Π. The natural addresses are represented on the y-axis, whereas the interleaved addresses are represented on the x-axis. Then the symbol with index x that is interleaved to the index $y = \Pi(x)$ is represented by a point with coordinates (x,y). Fig. 5 shows the representation of a random interleaver. On this figure, the shaded regions correspond to the regions where the points are associated with a consistency conflict. They are obtained by adding the temporal dimension resulting from schedule *SW-Σ*$^-$. Regions $\not\subset_{n\to i}$ and $\not\subset_{i\to n}$ are associated respectively with the transition $T_{n\to i}$ from the natural to interleaved half-iterations and with the transition $T_{i\to n}$ from the interleaved to the natural half-iterations.

It is assumed that the reading of the extrinsic inputs starts at time $\tau = 0$ and is performed window by window according to the backward order, *i.e.* in the decreasing order from N to 0. From time $\tau = L + l$, the extrinsic outputs are written into the memory window by window in the increasing order. If the first data read at time $\tau = 0$ (for the example of Fig. 5: $x = L$-1) corresponds to one of the data generated by the preceding half-iteration during its last Δ cycles (for the example of Fig. 5: $y > N - \Delta$), there is a consistency conflict. At time $\tau = \tau'$ the conflict region is reduced to Δ-τ' positions, and for $\tau = \Delta$, all symbols are conflict-free. This leads to the upper-triangle shape of $\not\in_{n\to i}$ in the first window associated with the transition $T_{n\to i}$. For transition $T_{i\to n}$,, the region $\not\in_{i\to n}$ also corresponds to an upper-triangle shape.

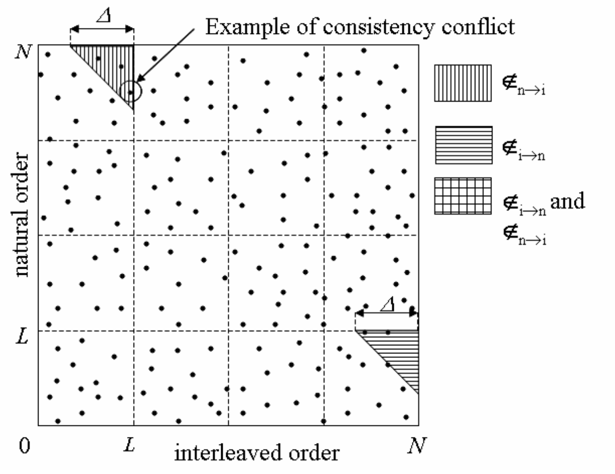


Fig. 5 Consistency conflicts regions.

Let us introduce the notion of uniform interleaver as follows. A uniform interleaver maps each symbol of the natural order to a given symbol of the interleaved order with a probability 1/*N*. Under this assumption, the proportion of consistency conflicts is given by the ratio of the area of the region $\not\in_{n\to I} \cup \not\in_{i\to n}$ divided by the total area of the space:

$$\eta(\Delta) = \frac{2\left(\Delta(\Delta+1)P^2/2\right)}{N^2} \approx \frac{\Delta^2 P^2}{N^2}. \qquad (8)$$

It can be seen that the proportion of consistency conflicts grows quadratically with Δ and P. Thus for high throughput architectures with large P there will be a large proportion of consistency conflicts.

4.2 Solving consistency conflicts

Similarly to the parallel conflicts mentioned in section 1, different techniques can be considered to resolve the issue of consistency conflicts. One example of execution-stage solution (involving using adhoc hardware) would be to use, instead of the "not-available" piece of data, a not up-to date version of it, namely its previous iteration version. Such a solution leads however to a suboptimal algorithm. Another alternative is proposed in the next section.

5. Joint interleaver / architecture design

In this section, we present a general methodology for avoiding consistency conflict by an adequate design of the interleaver matched to the decoding schedule. This methodology is illustrated on a simple interleaver construction enabling a significant activity increase.

5.1 Methodology

The interleaver design methodology is obtained by constraining the interleaver with "prohibited regions" corresponding to the overlapping parameter Δ. These constraints are modelled on the two dimensional representation of the interleaver by the prohibited regions $\not\in_{n\to i}$ and $\not\in_{i\to n}$ that define a mask for the interleaver, denoted $\in$. The constraint interleaver is then designed in order to satisfy the mask $\in$, *i.e.* no points $(x,\Pi(x))$ are inside the prohibited regions. If the interleaver design succeeds, there are by construction no conflicts in the Δ-overlapping architecture.

5.2 A simple example

In this example, we design a turbo code of size N that is decoded using P PUs performing schedule *OSW-Σ⁻*(Δ) with window size $L = M = N/P$. The corresponding interleaver mask is given in Fig. 6.a.

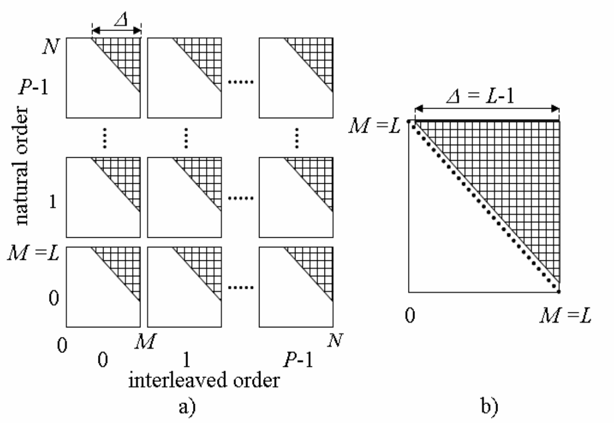


Fig. 6 Interleaver mask $\in$ for a turbo code with P PUs performing schedule *OSW-Σ⁻*(Δ): a) complete mask; b) partial mask for permutation Π_T and $\Delta = L$-1.

First, in order to guarantee the absence of parallel conflicts when accessing the memory banks in parallel, a specific parallel interleaver is used. This parallel interleaver is said to be "hierarchic" because it is designed with two levels of permutations: a first level (permutation Π_S) exchanges the data between the sub-blocks of M symbols and a second level (permutation Π_T) shuffles the data within each sub-block. A similar interleaver structure has also been used in [10] and [4] for solving parallel conflicts. The equation of the interleaver is given by the following equation:

$$\Pi(x) = M \cdot \Pi_S(x) + \Pi_T(x \bmod M). \qquad (9)$$

where Π_S is a circular rotation of equation

$$\Pi_S(x) = A(x \bmod P) + \lfloor x/M \rfloor \bmod P \qquad (10)$$

and A is bijection of 0,...,P-1 to 0,...,P-1 chosen at random. With this interleaver structure, the permutation Π_S resolves all parallel conflicts. Hence we use the permutation Π_T to solve consistency conflicts. The temporal permutation satisfying the mask is represented in Fig. 6.b for an overlapping depth $\Delta = L$-1.

For $\Delta \geq L$, the mask is said to be closed since no interleaver can satisfy it. For $\Delta = L$-1, a single permutation Π_T satisfy the mask: the reverse order where the symbol with index 0 is mapped to index M-1, index 1 to M-2, ..., M-1 to 0.

5.3 Results

Let us consider a double-binary turbo code of size $N = 1024$ double-binary symbols decoded by an architecture with parameters $P = 32$, $M = 32$, $L = 32$, $l = 8$. The turbo code is associated with a constrained hierarchical interleaver designed as described above. Its performance is compared to a conventional turbo code design using an ARP interleaver [9]. This latter is also decoded with P = 32 PUs but with schedule $SW\text{-}\Sigma^0$ of activity $\alpha_{SW-\Sigma^0} = 0.6$, higher than schedule $SW\text{-}\Sigma^-$. Using the schedule $OSW\text{-}\Sigma^-(L\text{-}1)$ for the proposed interleaver enables the activity to be increased from $\alpha_{SW-\Sigma^-} = 0.45$ to $\alpha_{OSW-\Sigma^-} = 0.8$, resulting in a significant increase in throughput. It is worth noting that the complexity of the corresponding architecture remains unchanged: $\Gamma_{OSW-\Sigma^-} = \Gamma_{SW-\Sigma^-}$.

On the one hand, the Frame Error Rate (FER) performance given in Fig. 7 shows that down to a FER of 10^{-2} (a practical FER for ARQ applications) the proposed solution does not suffer from performance degradation. On the other hand, the proposed solution enables the complexity of the decoder to be reduced by $1 - \Gamma_{OSW-\Sigma^-} / \Gamma_{SW-\Sigma^0} = 25$ %. And this, for an equivalent throughput (given by (1)) since, using (4)-(7) we have $\alpha_{OSW-\Sigma^-} \cdot \Gamma_{OSW-\Sigma^-} \approx \alpha_{SW-\Sigma^0} \cdot \Gamma_{SW-\Sigma^0}$. In this didactic example, where the interleaver has been highly constrained, the increase in the efficiency of the decoder (throughput divided by complexity) is paid by a degradation of the decoding performance for low FERs. Other trade-offs efficiency-performance can be achieved easily. In particular, for lower overlapping depth, and thus lower increase in efficiency, the constraints on the interleaver are relaxed, enabling this latter to be optimised for better performance at low FERs.

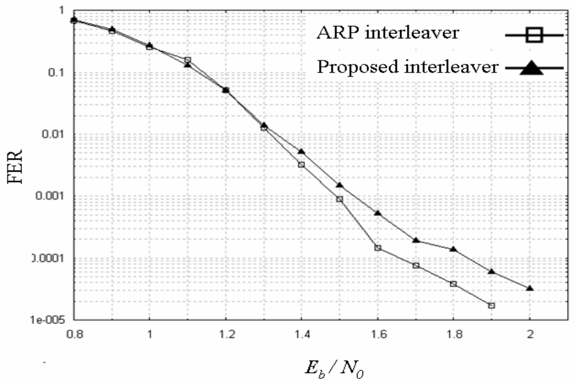


Fig. 7 FER performance of the proposed interleaver compared to a conventional ARP interleaver for 8 iterations of the Max-Log-MAP algorithm.

6. Conclusion

A new issue of parallel decoding of turbo codes has been addressed: how to use efficiently the computational resources? A general methodology for a joint interleaver-architecture design enabling a higher usage of these resources has been proposed. A simple example has been described enabling a typical complexity reduction of 25 % to be achieved easily. The proposed solution opens up a new direction for improvement of high-speed parallel turbo decoders. This approach is studied more deeply in [11], where maximal activity decoders associated with hierarchical interleavers are obtained in conjunction with Multiple Slice Turbo Codes [4].

7. References

[1] S. Benedetto *et al.*, “A soft-input soft-output maximum a posteriori (MAP) module to decode parallel and serial concatenated codes," JPL TDA Progress Report, Vol. 42-127, Nov. 1996.

[2] M.J. Thul, F. Gilbert, N. When, "Concurrent Interleaving architectures for high-throughput channel coding", ICASSP'03, Hong Kong, vol 2, pp 613-616, Apr. 2003.

[3] A. Tarable, S. Benedetto, G. Montorsi, "Mapping interleaving Laws to Parallel Turbo and LDPC decoder Architectures", IEEE Transactions on Information Theory, Vol. 50, No. 9, Sept. 2004.

[4] D. Gnaedig *et al.*, "On Multiple Slice Turbo Codes", Proc. 3rd Int. Symp. on Turbo Codes and Related Topics, Brest, pp.153-157, Sept. 2003.

[5] A. Dingninou, F. Raouafi et C. Berrou, "Organisation de la mémoire dans un turbo décodeur utilisant l'algorithme SUB-MAP", GRETSI '99, Vannes, pp. 71-74, Sept. 1999.

[6] E. Boutillon, W. J. Gross and P. G. Gulak, “VLSI Architectures for the MAP Algorithm,” IEEE Transactions on Communications, Vol. 51, No. 2, pp. 175-185, February 2003.

[7] C. Schurgers, F. Cathoor and M. Engels, "Memory Optimization of MAP Turbo Decoder Algorithms", IEEE Trans. on VLSI, vol.9, no. 2, Apr. 2001.

[8] Y. Wang *et al.*, "Reduced latency turbo decoding", Proc. 6th Workshop on Signal Proc. Advances in Wireless Com., pp. 930-934, June 2005.

[9] C. Berrou *et al.*, "Designing good permutations for turbo codes: towards a single model, " in Proc. ICC’04, Vol. 1, pp.341-345, June 2004.

[10] A. Nimbalker *et al.*, "Inter-Window Shuffle Interleavers for High Throughput Decoding," in Proc. 3rd Int. Symp. on Turbo Codes and Related Topics, Brest, pp.355-358, Sept. 2003.

[11] D. Gnaedig, “High Speed Decoding of Convolutional Turbo Codes”, PhD dissertation, June 2005.