High-efficiency Asymmetrical Shuffled decoding

Aomar Bourenane, Frédéric Guilloud, Matthieu Arzel, Alain Thomas

High-efficiency Asymmetrical Shuffled decoding

28 November - 1 December 2022
ESA/ESTEC, Noordwijk, The Netherlands

Aomar BOURENANE[(1)], Frederic GUILLOUD [(2)], Matthieu ARZEL[(2)], Alain THOMAS[(3)]

[(1)] Space and Communication Lab - Safran Data Systems / IMT Atlantique Lab-STICC, UMR CNRS 6285
5 Av. des Andes, 91940 Les Ulis, France.
aomar.bourenane@safrangroup.com

[(2)] IMT Atlantique Lab-STICC, UMR CNRS 6285
F-29238 Brest, France.
firstname.lastname@imt-atlantique.fr

[(3)] Space and Communication Lab Safran Data Systems
5 Av. des Andes, 91940 Les Ulis, France.
alain-dominique.thomas@safrangroup.com

INTRODUCTION

As an alternative for turbo-codes [1], S. Benedetto and G. Montesi have proposed Serial Concatenated Convolutional Codes (SCCC) [2]. While demonstrating a better error floor performance, their error-correcting performance is equivalent to its parallel counterpart and LDPC codes [3]. Additionally, SCCC encoders are appropriate for onboard satellite implementations thanks to their low complexity [4,5]. SCCC encoding comprises a serial concatenation of two convolutional encoders connected by an interleaver, the first corresponding to the outer code and the second to the inner code [2]. Hence, the iterative decoding technique employs Soft-Input Soft-Output (SISO) algorithms to iteratively decode the inner and the outer codewords while exchanging extrinsic information. Several efforts have been conducted to achieve high throughput turbo-code decoders in the literature using multiple parallelism techniques. Handful contributions are devoted to SCCC decoders, even though their efficient implementation is a prime concern for satellite telemetry systems [6]. Based on the sliding windows parallelism approach [7], [8] suggested a parallel SCCC decoder architecture with 16 concurrent SISO decoders. A high throughput Max-Log-Map SCCC turbo-decoder for an optical channel was presented in [9] for hardware implementation. Parallelism was implemented in both the calculation of state metrics inside SISO decoders and the use of sub-block parallelism. In these prior contributions, the parallelism incorporated in SCCC decoders was intended to accelerate the metric computations but not the exchange of extrinsic information, which is essential for increasing the decoder's throughput. In a previous paper, we introduced a new technique called Asymmetrical Shuffled decoding (ASD) that uses the shuffled decoding technique [10] to speed up and increase the exchange of extrinsic information [11]. The ASD technique uses the idle time of the hardware resources of the outer decoder to improve efficiency without adding any additional hardware resources by executing the outer component decoder twice in the same iteration. We are considering whether there is more parallelism or scheduling that we should explore to improve efficiency even more. A sub-block parallelism degree is applied to the ASD technique in this article to improve efficiency through the use of various scheduling techniques. This article offers two solutions that triple the decoder's overall throughput while preserving excellent efficiency. These designs use various parallelism techniques and proper scheduling.
The remaining sections are organized as follows. In Section 2, we describe the notations used throughout the study, as well as the encoding and decoding schemes of SCCC. In sections three and four, we suggest two high-efficiency ASD for systematic SCCC**.** The fifth section compares the various schemes based on their hardware and time resource usage, throughput, and performance. Finally, in section 6, we come to an end with a conclusion.

MODEL AND NOTATIONS

For clarity, we will refer throughout the study to the communication channel depicted in Fig.1, as a systematic SCCC encoder, a binary input memoryless additive white Gaussian noise (Bi-AWGN) channel, and an SCCC decoder.
The SCCC encoder has an outer convolutional recursive systematic code (RSC) $C^o$with rate $R^o = 1/2$, an interleaver Π, and an inner RSC C^i with rate $R^i = 1/2$. We assume that both component codes share the same constraint length K to simplify the notations without incurring generality. The superscripts i and o represent, respectively, the inner and outer codes. u_k denotes the k^{th} information bit at the input of the SCCC encoder, for $k \in [1 \,..\, N]$, which is encoded first by

the outer encoder. *Let* c_l^o for $l \in [0..2N]$ represents the outer coded bits that are interleaved into bits v_l and sent into the inner encoder. Let c_m^i for $m \in [0..4N]$ represent the inner coded bits modulated and sent through a Bi-AWGN channel. Let y_m represent the channel output proportional to the log-likelihood ratios (LLRs) of c_m^i, and let $\hat{u}_k$ represent the estimated information bit. Assuming that the outer (respectively inner) encoder is systematic, the output c_l^o (respectively c_m^i) is alternately equal to the systematic bit u_k (respectively v_l) and the redundant bit r_k^o (respectively r_l^i). Fig. 2. illustrates the functional block diagram of the SCCC decoder for two received sub-blocks. It is composed of three SISO decoders referred to as outer SISO, inner1 SISO, and inner2 SISO, as well as an interleaver and a deinterleaver. Serial concatenation of the two inner SISOs to the shared outer SISO is performed. Notably, even though the two inner SISOs share the same outer SISO but decode separately, the decoding process is similar to two serial turbo-decoders operating in tandem on two received sub-blocks without exchanging any pieces of information. The injection of channel information relative to the LLRs ($L1_c(v)$ and $L2_c(v)$) is highlighted in blue. This information is received at the receiver's input and is transmitted to the outer decoder through the blue wires. Using the same code rate as in the previous paper, we remind readers that the size of the frames to be decoded is inevitably different for the two component decoders (hence the decoding time is different). For example, at a code rate of ½ for the outer encoder, the inner decoder needs to decode twice the trellis length decoded by the outer decoder. The max-log-MAP algorithm [12] will be implemented as a SISO component to decrease computational complexity. Two decoding schemes are proposed based on the same decoding functional block of Fig. 2, except with two unique scheduling strategies to maximize decoder efficiency. In the following sections, these two decoding schemes are referred to as Asymmetrical Shuffled Simultaneous Decoding (ASSD), and Asymmetrical Shuffled Continuous Decoding (ASCD).

ASYMMETRICAL SHUFFLED SIMULTANEOUS DECODING SCHEME

Architectural Description

Since two inners are implemented, two sub-blocks can be decoded simultaneously, namely sub-block1 and sub-block2. Therefore, let us define two modes corresponding to their respective decoding: the Ping and the Pong modes.
The Ping (resp. Pong) mode: We associate this mode with the decoding of sub-block1 (resp. sub-block2). The inner1 (resp. inner2) decoder and the shared outer decoder are both decoding simultaneously. During this mode, the two SISOs are computing and exchanging the extrinsic information for the current trellis section through the two extrinsic memories

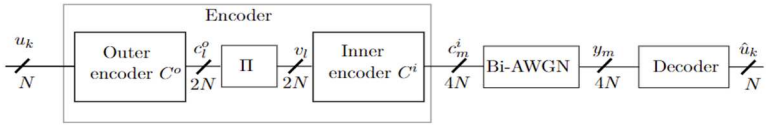

Fig. 1. SCCC encoder and transmission over a binary input AWGN (Bi-AWGN) channel model.

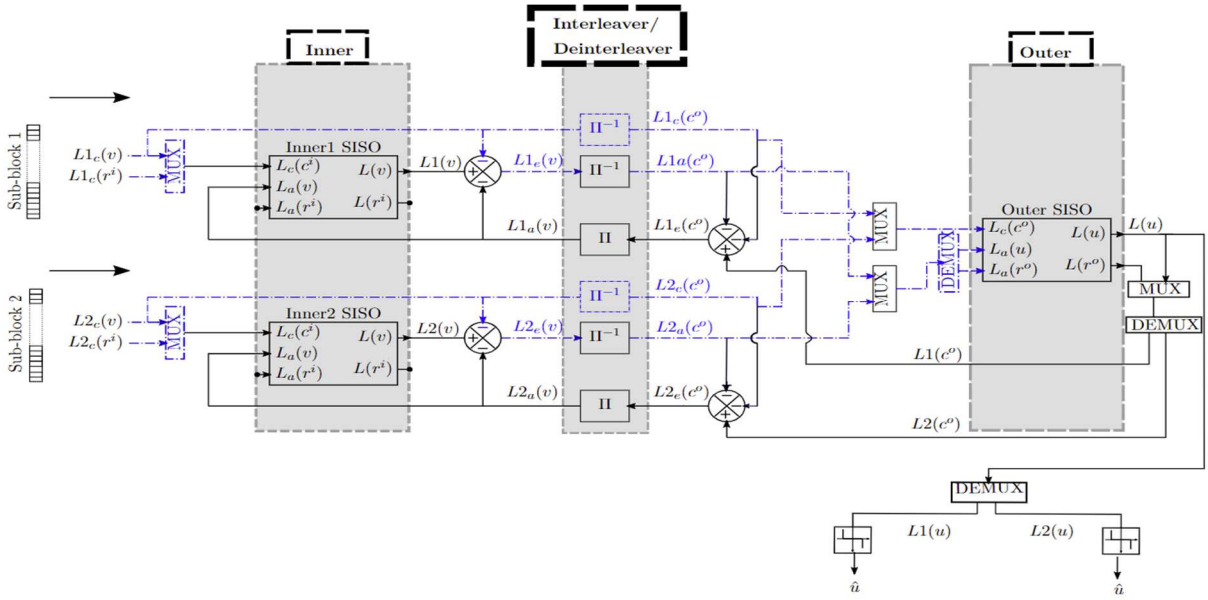

Fig. 2. Functional block diagram of the SCCC decoder for two received sub-blocks, where information (LLR) related to the channel observation is colored in blue.

(reading and writing). The hardware architecture of the proposed ASSD scheme is depicted in Fig.3. Three SISOs are employed to compute the state metrics (α, β) in parallel. The two upper SISOs correspond to the inner1 and inner2 decoders, whereas the third SISO corresponds to the shared outer decoder. Each inner SISO has its own memory for state metrics and extrinsic memory. The shared outer SISO is equipped with two state metrics and two extrinsic memories to accommodate the two Ping and Pong modes. Numerous multiplexers and demultiplexers (MUX, DEMUX) are employed to drive and supply the SISOs with the appropriate information. The control unit coordinates these modes.
In the baseline implementation, a single SISO component is deployed, and resources are utilized alternatively to perform both the inner and outer decoding. However, in the ASSD approach, three SISOs operate concurrently, each SISO processing one trellis section per clock cycle. The control unit schedules computations and drives memory accesses based on the decoded codeword. Resources must be sized to support the greedier case in a baseline scheme since they are shared. In an ASSD scheme, however, each resource is dedicated to a single SISO with the bare minimum hardware requirements, and thus the use of these resources is optimized for decoding time. However, the state metrics and extrinsic information memories are duplicated in the ASSD, as shown in Fig.3. Due to the fact that we duplicate all extrinsic information memories, this solution is suboptimal in terms of extrinsic memory use. However, it avoids memory access conflicts [13].

Core Operation And Decoding Time

After outlining the proposed hardware scheme for the ASSD decoding technique and highlighting its difference from the baseline scheme, in this section, we explain its basic operation.
The ASSD BCJR computation with a replica butterfly scheme is depicted in Fig.4. with three component decoders (inner1 scheduling (a), outer scheduling (b), and inner2 scheduling (c)). The trellis sections corresponding to each associated sub-block are depicted on the y-axis. The x-axis represents the time domain. The forward and backward recursion generation, as well as the extrinsic information, are marked by dotted-dashed lines. We can see that each inner decoder is processing a distinct $2N$ trellis section associated with the two modes' respective sub-blocks. On the other hand, the outer decoder processes two distinct N trellis sections associated with the two modes concurrently in the time domain, one section each clock cycle (circle-line marks symbolize the discretization). The grey area in each replica butterfly scheme symbolizes the memory allocation of each trellis section's state metrics across time. The asymmetry between the inner and outer decoders is expressed by the trellis length ratio.
During the ASSD iterative decoding, two SISOs play the role of two inner decoders decoding two independent sub-blocks, and the third SISO is shared between the two inner SISOs. It plays the role of a shared outer. The sharing is done in the time domain. As the two inner decoders need twice as much decoding time as the outer decoder (hence the term Asymmetrical in the word ASSD), we take advantage of the free time of the outer decoder to share it between the two inner decoders. This time-sharing is done by clock cycle, as shown in Fig.4. The outer decodes both sub-blocks, alternately, one trellis section per clock cycle per sub-block. The term "simultaneous" in the acronym ASSD comes from the fact that decoding the two sub-blocks is done simultaneously in a time-division manner, with two inner trellis sections in parallel.
A complete iteration of the proposed ASSD requires an amount of time T per two decoding sub-blocks of the two modes, as shown in Fig. 4. This means that the ASSD scheme requires a duration of $T/2$ per iteration to decode a received sub-block, when the baseline implementation with a SISO butterfly scheduling runs a complete iteration within an amount of time equal to $3T/2$ as we explained in [11]. Note that a replica butterfly scheme [14] is implemented to increase the extrinsic information exchange speed and the decoding throughput.

Memory Resources

In this research, we took into account the estimation of the state metrics and the extrinsic information memories. To simplify the resource estimation both in terms of memory and of computing units, we assume hereafter that the LLRs and the forward-backward metrics (α, β) are quantized with the same bit-width. The amount of memory required to save each state metric is given by the number of states per trellis section, *i.e.* 2^{K-1}, times the number of sections plus one, that is $2N + 1$ for the two inner codes and $N + 1$ for the shared outer code. Thus, the memory words required to store α and β state metrics for the two inner component decoders are specified by (1). In the case of the outer decoder, the number of memory words required to store the state metrics is given by (2). Due to the fact that we must process the decoding of two modes concurrently, we must double the size of the outer decoder metrics memory.

$$2 \times \left(2^{K}(2N + 1)\right) \quad (1)$$

$$2 \times \left(2^{K}(N + 1)\right) \quad (2)$$

For the baseline scheme, the largest trellis is the inner one and requires $2^{K}(2N + 1)$ memory words to be saved. The total state metrics memory needed by the ASSD scheme is given by (3).

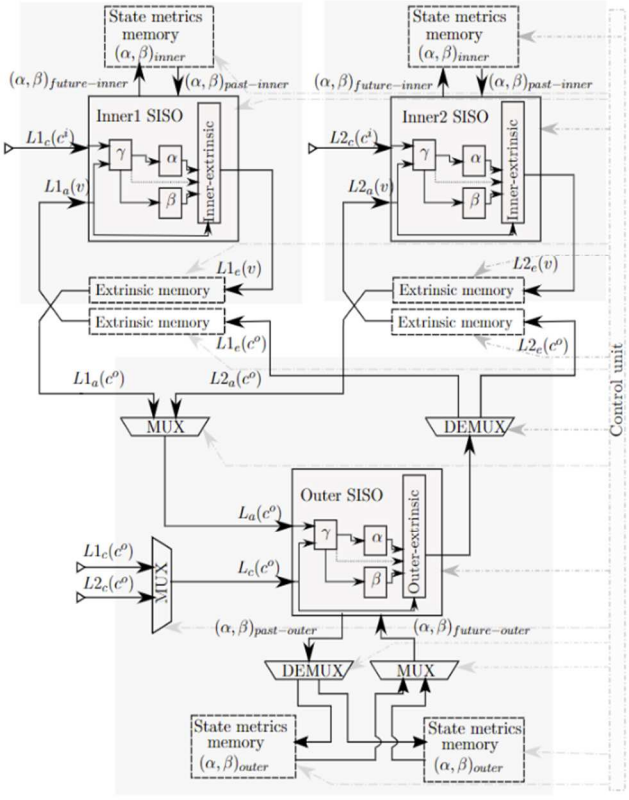

Fig.3. ASSD proposed scheme

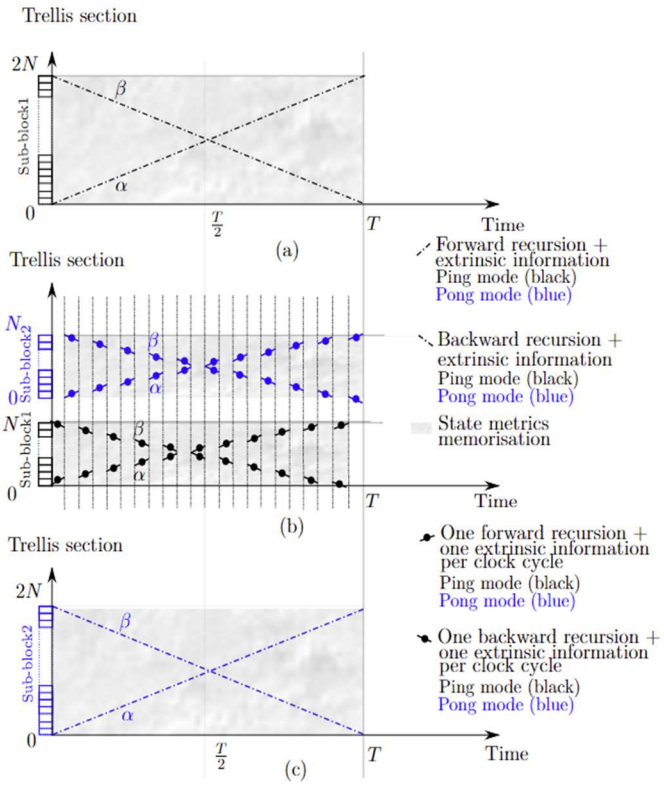

Fig.4. ASSD BCJR computation with replica butterfly scheme: (a) inner1 decoder scheduling, (b) outer decoder scheduling, (c) inner2 decoder scheduling.

$$2 \times \left(2^K\,(2N+1) + 2^K(N+1)\right) = 2^{K+1}(3N+2) \tag{3}$$

The amount of memory required to save extrinsic information in the inner decoder is related to the number of systematic bits $2N$. In the outer decoder, it is given by the number of the coded bits, which is $2N$ because the extrinsic information of both the systematic and redundant bits must be saved. Notably, the extrinsic information memory needs for the ASSD scheme are duplicated to allow memory to be accessed concurrently by the inner and outer decoders, as illustrated in Fig.4. The figure depicts the presence of four memories for extrinsic information. As a result, the circuit consumes a total of $8N$ memory words for the extrinsic information. The total amount of memory words used in the ASSD scheme is indicated by the variable M_{ASSD}, which equals

$$M_{ASSD} = 2^{K+1}(3N+2) + 8N \tag{4}$$

Only one extrinsic memory of $2N$ words is required in the baseline case. The total number of memory words used in the Baseline Decoding (BD) scheme is indicated by the variable M_{BD} and is equal to

$$M_{BD} = 2^K(2N+1) + 2N \tag{5}$$

Computing Resources

This section reports the estimation of the ASSD technique's overall computing resource requirements in order to compare it to the baseline technique and calculate its efficiency. We consider the amount of resources required to compute the state metrics (α,β) and extrinsic information for the two inner component decoders $(Cr_{c_1^i}, Cr_{c_2^i})$ and the shared outer component decoder Cr_c^o in a single trellis section. Calculation of state metrics in a trellis section with 2^{K-1} states is accomplished by the use of 2^{K-1} Add-Compare-Select (ACS) units for α or β state metrics, i.e., 2×2^K ACS, for the two inner decoders and 2^K for the outer since we assume the same constraint length K for the three-component codes.
Extrinsic information is obtained by first adding the two state metrics to the branch metrics, which needs two ADD (Addition operator) per branch, thus $2 \times 2^K = 2^{K+1}$ ADD per trellis section. Then, for the inner decoder (extrinsic information on the systematic bits), the maximum sum over all branches related to a systematic bit equal to 0 must be chosen, as well as the maximum sum over all branches related to a systematic bit equal to 1. Finding the maximum over $2^K/2$ values requires $2^{K-1} - 1$ Compare-and-Select (CS) units, resulting in a total of $2^K - 2$ CS. Finally, the subtraction

between the two preceding maximums and by subtracting the a priori information and the channel information from the result (cost of 3 ADD), the extrinsic information for the systematic bit on the considered trellis section of the inner decoder will be obtained. The two inner component decoders are identical, so the hardware requirements are equal. Each inner trellis calculation requires 2^KACS, $2^{K+1}+3$ ADD and 2^K-2 CS. If one uses the equivalences: an ACS is 2 additions, a subtraction (equivalent to an addition) and a selection S (3ADD+S), a CS is an addtion and a selection S (ADD+S), we obtain the following results

$$Cr_{c_1^i} = Cr_{c_2^i} = (3 \times 2^{K+1} + 1)ADD + (2^{K+1} - 2)S \quad (6)$$

As for the shared outer decoder, the calculation is the same, except that two extrinsic information values have to be calculated for each trellis section: one for the systematic bit and one for the redundant bit. Hence the number of operations dedicated to the inner extrinsic information is doubled, yielding a total of

$$Cr_c^o = (9 \times 2^K + 2)ADD + (3 \times 2^K - 4)S \quad (7)$$

Finally, the overall complexity of the proposed ASSD scheme is obtained by adding the complexity of the two inner and outer decoders, resulting in

$$Cr_{ASSD} = Cr_c^o + Cr_{c_1^i} + Cr_{c_2^i} = (21 \times 2^K + 4)ADD + (7 \times 2^K - 8)S \quad (8)$$

Whereas for the baseline scheme, the complexity is the one of the outer decoder

$$Cr_{BD} = Cr_c^o = (9 \times 2^K + 2)ADD + (3 \times 2^K - 4)S \quad (9)$$

ASYMMETRICAL SHUFFLED CONTINUOUS DECODING SCHEME

Architectural Description

The proposed ASCD scheme's hardware architecture is depicted in Fig.5. Three SISOs are used in parallel to compute the state metrics (α, β). The two upper SISOs represent the inner1 and inner2 decoders, respectively, while the third SISO represents the shared outer decoder. Each inner SISO has its own extrinsic and state metrics memory. Compared to the ASSD scheme, which uses two state metrics memories, the shared outer SISO scheme uses a single state metrics memory shared (drawn in blue in the figure) between the two *Ping-Pong* modes. Additionally, the shared outer component decoder includes two extrinsic memories to support the *Ping* and *Pong* modes (one extrinsic memory allocated to the decoding time of each mode). Numerous multiplexers and demultiplexers (MUX, DEMUX) are used to drive and supply information to the SISOs. As illustrated in Fig.5, the control unit coordinates these modes. As is the case with ASSD, the ASCD scheme is based on shuffled decoding, appropriate scheduling, and sub-block parallelism. The extrinsic information memories are duplicated, this technique is sub-optimal in terms of extrinsic memory consumption, but it eliminates memory access conflicts. Unlike the ASSD scheme, the ASCD scheme is more efficient in memory savings by using a single state metrics memory to support the two modes, Indeed, state metrics are very greedy in memory resources, especially when the number of code states and/or the sub-block trellis length are significant (in the range of $2^{K-1} \times (N+1)$).

Core Operation And Decoding Time

This section discusses the technique's essential operation. The outer decodes both sub-blocks alternately, once every half iteration. The term "Continuous" in the word ASCD refers to the fact that the two sub-blocks are decoded sequentially in a continuous manner, including all the trellis sections of the first sub-block of the *Ping* mode being decoded in the first half iteration of the iterative decoding and then all the trellis sections of the second sub-block of the *Pong* mode being decoded in the second half iteration of the iterative decoding. This method is performed continuously (*Ping-> Pong->Ping...*) until the total number of iterations of iterative decoding is reached. The ASCD BCJR computation with a replica butterfly scheme is depicted in Fig.6, with three component decoders (inner1 scheduling (a), outer scheduling (b), and inner2 scheduling (c)). The trellis sections corresponding to each associated sub-block are depicted on the y-axis. The x-axis represents the time domain. The forward and backward recursion generation, as well as the extrinsic information, are marked by dotted-dashed lines. We can see that each inner decoder is processing a distinct $2N$ trellis section associated with the two modes' respective sub-blocks. On the other hand, the outer decoder processes two distinct N trellis sections associated with the two modes sequentially in a continuous manner in the time domain. The grey area in each replica butterfly scheme symbolizes the memory allocation of each trellis section's state metrics across time. The asymmetry between the inner and outer decoders is expressed by the trellis size ratio. A complete iteration of the proposed ASCD requires an amount of time T per two decoding sub-blocks, as shown in Fig.6: an amount of time of T for each inner component decoder, an amount of time of $T/2$ for each sub-block of each decoding mode for the shared outer decoder. Because of the fact that the three-component decoders are working concurrently, the total amount of time required for decoding the two sub-blocks is T. Thus, the decoding time required per sub-block, per iteration is $T/2$.

Memory Resources

Equation (10) specifies the memory words required to store the α and β state metrics for the two inner component decoders. Likewise, the number of memory words required by the outer decoder to store the state metrics is specified by (11). We do not need to double the size of the outer decoder metrics memory as we did for the ASSD scheme since we release the allocation of this memory to one of the modes every half iteration.

$$2 \times \left(2^K\,(2N+1)\right) \tag{10}$$

$$2^K(N+1) \tag{11}$$

The total amount of memory words used in the ASCD scheme is indicated by the variable M_{ASCD}, which equals

$$M_{ASCD} = 2^K(5N+3) + 8N \tag{12}$$

It is interesting to mention that it can be seen by comparing Fig.(4)(b) and Fig.6 (b) that the state metrics memory area used by the shared outer decoder in the case of the ASCD scheme is half the one used for the ASSD scheme.

Computing Resources

This section reports the estimation of the ASCD technique's overall computing resource requirements in order to compare it to the baseline technique and calculate its efficiency. The ASCD scheme has the same needs of computing resources than the ASSD scheme,

$$Cr_{ASCD} = Cr_{ASSD} = Cr_c^o + Cr_{c_1^i} + Cr_{c_2^i} = (21 \times 2^K + 4)ADD + (7 \times 2^K - 8)S \tag{13}$$

PERFORMANCE AND COMPARISON

In order to estimate the Bit Error Rate performance (BER) of the two proposed schemes, the received frames of $N = 4320$ are divided into $P = 120$ sub-blocks and decoded in parallel. Each sub-block requires the initialization of recursion metrics, as only frame ends and not sub-block ends include recursion metric information either by using a circular trellis [15] or through tail-biting [16]. The first technique entails estimating the initiation of sub-blocks' limits based on the pre-calculations of neighboring blocks called acquisition (ACQ)-based initialization [17]. The second technique is referred to as initialization by message forwarding or next iteration initialization (NII) [18]. Two methods (ACQ-NII) can be combined in the third technique as proposed in [19]. In this paper, only the NII technique is considered.

We consider that each iteration of the proposed ASSD (resp.ASCD) scheme lasts for a predetermined time T. In comparison, one iteration of the BD scheme takes $3T/2$, and because the two proposed schemes process two modes concurrently (thus, two sub-blocks are decoded), the proposed scheme's decoding time is halved. Therefore, the total decoding time for 9 iterations is stated using (14) for the two proposed schemes and (15) for the BD scheme.

$$T_{ASSD\ or\ ASCD} = \frac{T}{2} \times 9 = 4.5T \tag{14}$$

$$T_{BD} = \frac{3T}{2} \times 9 = 13.5T \tag{15}$$

As expressed by the time ratio (16), the two schemes achieve a threefold increase in throughput over the BD scheme.

$$\frac{T_{BD}}{T_{ASSD\ or\ ASCD}} = 3 \tag{16}$$

In Fig.7 (resp. Fig.8), the ASSD (resp. ASCD) and the BD schemes are represented in terms of BER as a function of signal-to-noise ratio, and compared to the BD and the ASSD (resp. ASCD) schemes without sub-block technique.

In this simulation, the iterative decoding is conducted utilizing a total of 9 iterations. According to the two figures, the increase in throughput has no impact on the BER performance of the ASSD scheme and has only a slight degradation of 0.09 dB in the case of the ASCD scheme. Therefore, the BER performance of the two schemes is comparable.

Considered numerical values (with $K = 3$ and $N = 4320$) for ASD, ASSD, and ASCD schemes in comparison to the BD scheme are summarized in Table 1. The table considers several criteria, including a fixed BER at 10^{-5} for the comparison, the considered total number of iterations, SISO computing time, the number of memory words stored during the decoding process, addition operations (ADD), the number of used selectors, and finally, the throughput ratio. The percentages in green (resp. red) represent the relative improvement (resp. degradation) of hardware resources compared to the BD scheme. The percentages are measured using the efficiency metric described in [11] to answer the question whether the increase in throughput results in an identical increase in hardware resources

Clearly, this faster convergence and higher throughput come at the expense of additional hardware resources. However, the table demonstrates unequivocally that the three proposed decoding techniques need reduced hardware resources for an equivalent throughput, moreover: Both ASSD and ASCD decoding schemes offer three times higher throughput than the BD scheme, and **50%** more throughput than the ASD scheme. The ASD scheme is the best plan for conserving

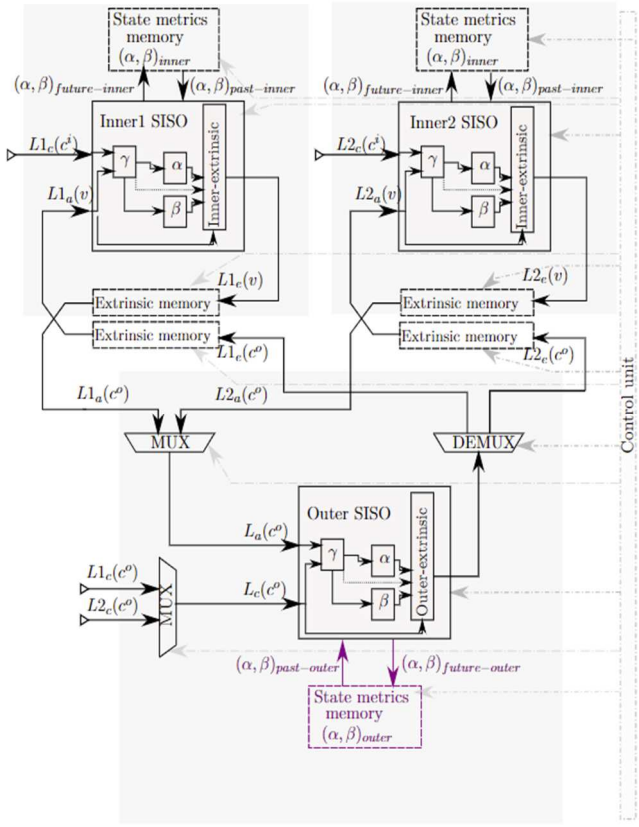

Fig.5. ASCD proposed scheme

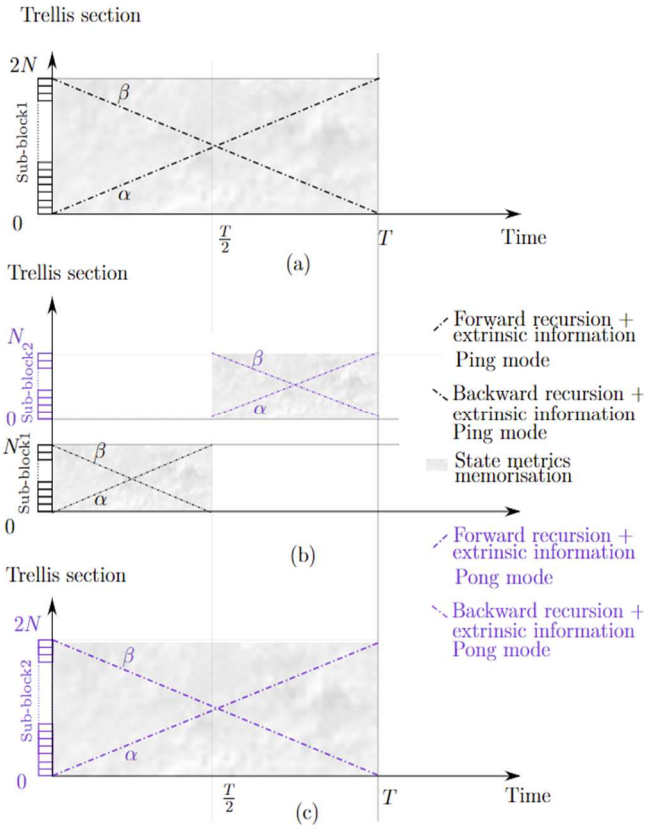

Fig.6.ASCD BCJR computation with replica butterfly scheme: (a) inner1 decoder scheduling, (b) outer decoder scheduling, (c) inner2 decoder scheduling.

memory resources, 29% compared to 12% for the ASCD scheme. Even though it offers less throughput than ASCD scheme. Therefore, the ASD scheme may be the best choice for an architecture with limited memory resources. Both the ASSD and the ASCD schemes are the best plans for conserving computational resources. Furthermore, the ASCD scheme offers memory resource savings, making it the superior alternative.

CONCLUSION

In this article, we propose two different schemes that offer better computing efficiencies and triple the overall throughput of the decoder compared to the baseline scheme. The two schemes use different types of parallelism: Shuffled decoding and Sub-block parallelism. The ASSD scheme allows for 50% more in throughput than the ASD scheme presented in [11] and better hardware computing efficiency. In addition, we proposed the ASCD scheme, which is based on relevant scheduling that offers the ASSD scheme's advantages and better memory efficiency, making it the superior alternative.

REFERENCES

Table 1. Considered numerical values for the BD vs ASD vs ASCD vs ASSD schemes. In green (resp. red) the saving (resp. adding) hardware resources normalized to the BD scheme.

Criteria	Baseline scheme	ASD scheme	ASSD scheme	ASCD scheme
BER at 10^{-5}	1.01 dB	1.01 dB	1.01 dB	1.1 dB
Number of iterations	8	6	8	8
SISO processing time	$\frac{3T}{2}$	T	$\frac{T}{2}$	$\frac{T}{2}$
# memory words	77768	120976 ↓ 29%	241952 ↑ 4%	207384 ↓ 12%
# ADD	74	123 ↓ 20%	172 ↓ 29%	172 ↓ 29%
# S	20	34 ↓ 18%	48 ↓ 25%	48 ↓ 25%
Throughput ratio	100%	200%	300%	300%

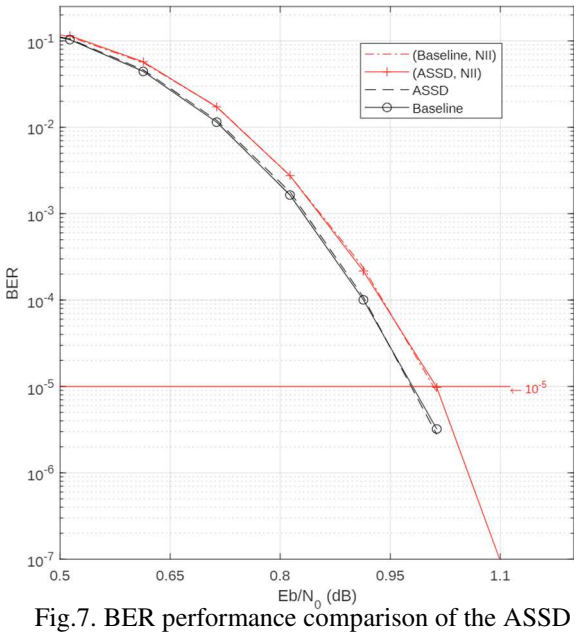

Fig.7. BER performance comparison of the ASSD and baseline schemes as a function of E_b/N_0.

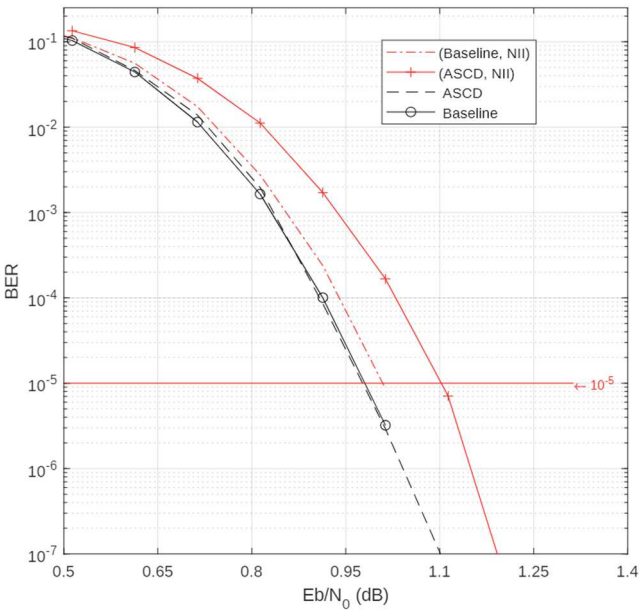

Fig.8. BER performance comparison of the ASCD and baseline schemes as a function of E_b/N_0.

[1] Berrou, C., & Glavieux, A. (1996). Near optimum error correcting coding and decoding: Turbo-codes. *IEEE Transactions on communications*, *44*(10), 1261-1271.

[2] Benedetto, S. (1996). Serial concatenation of block and convolutional codes. *Electron. Lett.*, *32*(10), 887-888.

[3] Gallager, R. (1962). Low-density parity-check codes. *IRE Transactions on information theory*, *8*(1), 21-28.

[4] Benedetto, S., Divsalar, D., Montorsi, G., & Pollara, F. "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding." *IEEE Transactions on information Theory* 44.3 (1998): 909-926.

[5] Amat, A. G., Montorsi, G., & Vatta, F. (2009). Design and performance analysis of a new class of rate compatible serially concatenated convolutional codes. *IEEE transactions on communications*, *57*(8), 2280-2289.

[6] Bertolucci, M., Falaschi, F., Cassettari, R., Davalle, D., & Fanucci, L. "A comprehensive trade-off analysis on the CCSDS 131.2-B-1 extended modcod (SCCC-X) implementation." *2020 23rd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2020.

[7] Schurgers, C., Catthoor, F., & Engels, M. (2001). Memory optimization of MAP turbo decoder algorithms. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *9*(2), 305-312.

[8] Martina, M., Molino, A., Vacca, F., Masera, G., & Montorsi, G. (2006). High throughput implementation of an adaptive serial concatenation turbo decoder. *Journal of Communications Software and Systems*, *2*(3), 252-261.

[9] Shoup, R. (2006, August). Hardware implementation of a high-throughput 64-PPM serial concatenated turbo decoder. In *Optical Information Systems IV* (Vol. 6311, pp. 250-257). SPIE.

[10] Zhang, J., & Fossorier, M. P. (2005). Shuffled iterative decoding. *IEEE Transactions on Communications*, *53*(2), 209-213.

[11] Bourenane, A., Arzel, M., Guilloud, F., & Thomas, A. (2021). Shuffled Decoding of Serial Concatenated Convolutional Codes. In *2021 11th International Symposium on Topics in Coding (ISTC)* (pp. 1-5). IEEE.

[12] Koch, W., & Baier, A. (1990, December). Optimum and sub-optimum detection of coded data disturbed by time-varying intersymbol interference (applicable to digital mobile radio receivers). In *[Proceedings] GLOBECOM'90: IEEE Global Telecommunications Conference and Exhibition* (pp. 1679-1684). IEEE.

[13] Giulietti, A., Van der Perre, L., & Strum, M. (2002). Parallel turbo coding interleavers: avoiding collisions in accesses to storage elements. *Electronics letters*, *38*(5), 1.

[14] Zhang, J., Wang, Y., Fossorier, M., & Yedidia, J. S. (2005, September). Replica shuffled iterative decoding. In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.* (pp. 454-458). IEEE.

[15] Ma, H., & Wolf, J. (1986). On tail biting convolutional codes. *IEEE Trans on Communications*, *34*(2), 104-111.

[16] C. Berrou, "Codes et turbocodes," 2007

[17] Sun, Y., & Cavallaro, J. R. (2008, October). Unified decoder architecture for LDPC/turbo codes. In *2008 IEEE Workshop on Signal Processing Systems* (pp. 13-18). IEEE.

[18] Dielissen, J. (2000). State vector reduction for initialization of sliding windows MAP. In *Proc. 23nd Int, Symp. Turbo Codes Related Topics, 2000* (pp. 387-390).

[19] May, M., Ilnseher, T., Wehn, N., & Raab, W. (2010, March). A 150Mbit/s 3GPP LTE turbo code decoder. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)* (pp. 1420-1425). IEEE.