Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2010, Article ID 927920, 11 pages
doi:10.1155/2010/927920

Research Article

Parallelism Efficiency in Convolutional Turbo Decoding

Olivier Muller,[1] Amer Baghdadi,[2] and Michel Jézéquel[2]

[1] TIMA Laboratory, INP Grenoble, 46 avenue Félix Viallet, 38031 Grenoble, France
[2] Institut Telecom, Telecom Bretagne, Université Européenne de Bretagne, UMR CNRS 3192 Lab-STICC, Technopôle Brest Iroise, CS 83818, 29238 Brest, France

Correspondence should be addressed to Amer Baghdadi, amer.baghdadi@telecom-bretagne.eu

Received 25 May 2010; Revised 20 October 2010; Accepted 21 November 2010

Academic Editor: Norbert Goertz

Parallel turbo decoding is becoming mandatory in order to achieve high throughput and to reduce latency, both crucial in emerging digital communication applications. This paper explores and analyzes parallelism techniques in convolutional turbo decoding with the BCJR algorithm. A three-level structured classification of parallelism techniques is proposed and discussed: *BCJR metric level parallelism*, *BCJR-SISO decoder level parallelism*, and *Turbo-decoder level parallelism*. The second level of this classification is thoroughly analyzed on the basis of parallelism efficiency criteria, since it offers the best tradeoff between achievable parallelism degree and area overhead. At this level, and for subblock parallelism, we illustrate how subblock initializations are more efficient with the message passing technique than with the acquisition approach. Besides, subblock parallelism becomes quite inefficient for high subblock parallelism degree. Conversely, component-decoder parallelism efficiency increases with subblock parallelism degree. This efficiency, moreover, depends on BCJR computation schemes and on propagation time. We show that component-decoder parallelism using shuffled decoding enables to maximize architecture efficiency and, hence, is well suited for hardware implementation of high throughput turbo decoder.

1. Introduction

Turbo decoding [1] is increasingly proposed in emerging and future digital communication systems, for example, fiber-optic communication, wireless communication, and storage applications. Practical turbo decoder designs, as the one used for IEEE 802.16e, Long-term Evolution (LTE) or IEEE 802.11 standards, require high data throughput (several hundred of Mbps) and low latency (ten ms or so). To cope with these requirements, turbo decoder implementations have to be massively parallel. Therefore, the parallelism involved in implementations of this iterative process has to be carefully analyzed through real industrial constraints such as area and throughput.

In iterative decoding algorithms, the underlying turbo principle relies on extrinsic information exchanges and iterative processing between different Soft-Input Soft-Output (SISO) modules. Using input information and *a priori* extrinsic information, each SISO module computes *a posteriori* extrinsic information. This constitutes the *a priori* information for the other modules and is exchanged via interleaving and deinterleaving processes. For convolutional turbo codes, the SISO modules process the BCJR or forward-backward algorithm [2] which is the optimal algorithm for the maximum a posteriori (MAP) decoding of convolutional codes. So, a BCJR SISO firstly computes branch metrics (or γ metrics), which represent the probability of a transition occurring between two trellis states. Then a BCJR SISO computes forward and backward recursions. Forward recursion (or α recursion) computes a trellis section (i.e., the probability of all states of the trellis) using the previous trellis section and branch metrics between these two sections, while backward recursion (or β recursion) computes a trellis section using the future trellis section and branch metrics between these two sections. Finally, extrinsic information is computed from the forward recursion, the backward recursion and the extrinsic part of the branch metrics. Therefore, turbo decoders, because of iterative decoding process and BCJR complexity (bidirectional recursive computing), imply optimal parallelism exploitation in order to achieve high-data rates required in present and future applications.

Parallelism in convolutional turbo decoding has been widely investigated in the literature over the last few years. At a fine-grain level, parallelism was investigated inside the BCJR algorithm on elementary computations [3, 4]. At a coarse grain level, explored parallelism techniques are based on frame decoding schemes. State-of-the-art research is mainly focused on parallel processing of frame subblocks and on the parallel processing issues, such as computation complexity [3, 5], memory saving [3, 5, 6], initializations [5, 7], or on-chip communication requirements [8, 9]. Recently, a new parallelism technique named shuffled decoding was introduced to process in parallel the component decoders [10, 11]. However, interactions between these diverse parallelism techniques and different granularity levels are rarely discussed. Thus, predicting the combination of these parallelism techniques, in order to reach optimal parallelism for given performance requirements, becomes a complex task of hardware implementation.

In this paper, we propose a three-level classification of existing parallelism techniques in convolutional turbo decoding with the BCJR algorithm and analyze more thoroughly the second level which includes subblock parallelism and component-decoder parallelism. Performance analyses of these techniques are conducted separately on the basis of parallelism efficiency criteria. Then, efficiency of combined parallelism techniques is improved collectively by taking into account interactions between these parallelism techniques.

The rest of the paper is organized as follows. The next section presents definitions of parallelism metrics that will be used in the paper. Section 3 analyzes all parallel processing techniques of turbo decoding and proposes a three-level classification of these techniques. In Sections 4 and 5, subblock parallelism and component-decoder parallelism (shuffled decoding) are, respectively, analyzed on the basis of parallelism efficiency criteria. Finally, Section 6 summarizes the obtained results and concludes the paper.

2. Definitions of Parallelism Metrics

Parallelism corresponds to the simultaneous execution of several independent processes in order to reduce processing time. Thus, due to dependencies in a real system, parallelism exploitation is naturally limited. In practice, algorithms always contain a sequential and incompressible part (e.g., processing synchronization) and a parallelizable part. The speedup $S(d)$ of an algorithm is defined by

$$S(d) = \frac{t(1)}{t(d)} = \frac{1}{r_s + r_p/d}, \tag{1}$$

where $t(1)$ is the execution time of the algorithm without parallelism and $t(d)$ its execution time using a parallelism degree d. The speedup generally follows the well-known Amdahl's law [12], where r_s is the sequential part ratio and r_p the parallelizable part ratio (such as $r_s + r_p = 1$)

For on-chip implementation purpose, parallelism has to be evaluated according to other criterion than speedup (e.g., accuracy, flexibility, latency, consumption, and chip area). In current implementations, the most important criterions are throughput and chip area. Therefore, in this paper, the efficiency of convolutional turbo decoder architectures, denoted M_e, is evaluated through execution time t and chip area C for equivalent results in terms of error-rates

$$M_e = \frac{1}{tC}. \tag{2}$$

Let $E(d)$ be the efficiency of a parallelism defined as

$$E(d) = \frac{M_e(d)}{M_e(1)} = \frac{t(1)C(1)}{t(d)C(d)} = \frac{S(d)}{O(d)}, \tag{3}$$

where $M_e(d)$ is the efficiency of a system using a parallelism degree d for the studied parallelism and $M_e(1)$ its efficiency without this parallelism. This metric can also be seen as the ratio between decoding speedup of a parallelism and its area overhead $O(d) = C(d)/C(1)$, where $C(d)$ (resp. $C(1)$) is the chip area of the system using a parallelism degree d (resp. without parallelism).

Since parallelism implies the duplication of only one part of the architecture (the ratio of the duplicated part in a nonparallelized chip area is denoted r_{dup}), area overhead can be rewritten

$$O(d) = 1 + r_{\text{dup}}(d-1). \tag{4}$$

3. Parallel Processing Levels

In turbo decoding with the BCJR algorithm, parallelism techniques can be classified at three levels: *BCJR metric level parallelism, BCJR SISO decoder level parallelism,* and *Turbo-decoder level parallelism*. The first (lowest) parallelism level concerns symbol elementary computations inside an SISO decoder processing the BCJR algorithm. Parallelism between these SISO decoders, inside one turbo decoder, belongs to the second parallelism level. The third (highest) parallelism level duplicates the turbo decoder itself.

3.1. BCJR Metric Level Parallelism

3.1. BCJR Metric Level Parallelism. The BCJR metric level parallelism concerns the processing of all metrics involved in the decoding of each received symbol inside a BCJR SISO decoder. It exploits the inherent parallelism of the trellis structure [3, 4], and also the parallelism of BCJR computations [3–5].

3.1.1. Parallelism of Trellis Transitions. Trellis-transition parallelism can easily be extracted from trellis structure as the same operations are repeated for all transition pairs. In log-domain [13], these operations are either ACS operations (Add-Compare-Select) for the max-log-MAP algorithm or ACSO operations (ACS with a correction offset [13]) for the log-MAP algorithm.

Each BCJR computation requires a number of ACS-like operations equal to half the number of transitions per trellis section. Thus, this number, which depends on the structure of the convolutional code, constitutes the upper bound of the trellis-transition parallelism degree.

Furthermore this parallelism implies low area overhead (as only the ACS units have to be duplicated). In particular, no additional memories are required since all parallelized operations are executed on the same trellis section and, in consequence, on the same data.

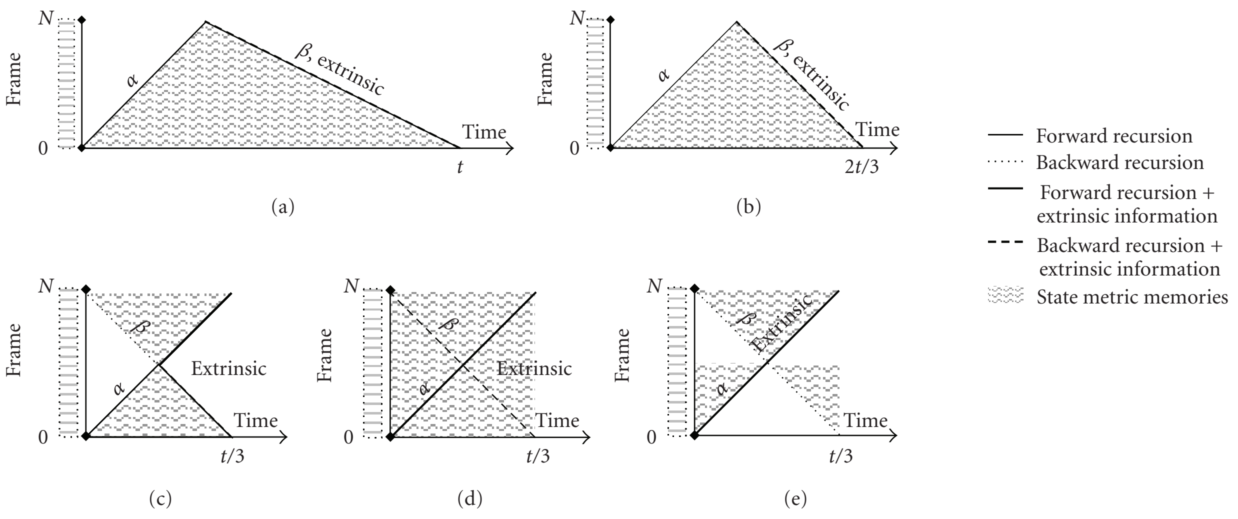

Figure 1: BCJR computation schemes: (a) forward-backward without parallelism, (b) original forward-backward, (c) butterfly, (d) replica butterfly, (e) forward butterfly.

3.1.2. Parallelism of BCJR Computations. A second metric parallelism can be orthogonally extracted from the BCJR algorithm through a parallel execution of the three BCJR computations (forward recursion, backward recursion, and extrinsic information) [3, 5]. To preserve data dependencies in the BCJR algorithm, computations are scheduled following particular schemes as depicted in Figure 1. On the depicted schemes, use of state metric memories is represented by the gray regions. Size of this memory is equal to maximum height of gray region during the decoding. State memory requirement can be reduced using sliding window method [6, 14].

Parallel computation of backward recursion and extrinsic information was proposed with the original forward-backward scheme [2], depicted in Figure 1(b) In this scheme, we can notice that BCJR computation parallelism degree is equal to one in the forward part and two in the backward part. Thus, it enables a speedup of 1.5 in comparison with the sequential BCJR algorithm (BCJR computation parallelism degree is always one as depicted in Figure 1(a)) and its state metric memory depth is equal to frame length. To increase this parallelism degree, several schemes were proposed [5]. The most common is the butterfly scheme (Figure 1(c)) which doubles the parallelism degree of the original scheme through the parallelism between the forward and backward recursion computations (degree 2 in the first half of the butterfly and degree 4 in the second half) in order to perform a speedup of 3. Various schemes have been derived from this butterfly scheme. For example, the replica butterfly scheme (Figure 1(d)) extends the extrinsic information computations to the first half of the butterfly. This scheme was proposed in [15] to improve the shuffled decoding convergence, but it requires four BCJR computation units (full-time usage for a speedup of 3) and state metric storage between two consecutive iterations (since the extrinsic information computations in the first half of the butterfly require one recursion computation (forward or backward) and take the other one (backward or forward) from the state metric storage). Globally, memory size is the same with this scheme as with the butterfly scheme or forward-backward one. A further example is the forward butterfly scheme (Figure 1(e)) that performs extrinsic information computation only in forward direction. Consequently, its BCJR computation parallelism degree is three on both half of the butterfly. Furthermore, the state metric memory is twice less deeper in this case. The resulting BCJR-SISO decoder requires smaller area (typically 30% less in comparison with others butterfly schemes), but the iterative process has a slower convergence (see Section 4). We can note that, with all these schemes, BCJR computation parallelism is performed without any memory increase and only BCJR computation resources have to be duplicated.

As a conclusion, BCJR metric level parallelism (trellis-transition parallelism and BCJR computation parallelism) induces a minimal area overhead as it does not affect memory size, which occupies most of the area in a turbo decoder circuit. Nevertheless the parallelism degree is limited by the code structure and the decoding algorithm. Thus, achieving higher parallelism degree implies exploring higher processing levels.

3.2. BCJR-SISO Decoder Level Parallelism. The second level of parallelism concerns the SISO decoder level. It consists in using multiple SISO decoders, each executing the BCJR algorithm and processing a subblock of the same frame in one of the two component decoders. At this level, parallelism can be applied either on subblocks and/or on component decoders.

3.2.1. Subblock Parallelism. In subblock parallelism, each frame is divided into d_{SB} subblocks and then each subblock is processed on a BCJR-SISO decoder using adequate initializations [6, 16, 17]. In fact, only two different initialization methods, namely acquisition and message passing (also know as next-iteration initialization) exist and are analyzed in Section 4.

Besides duplication of BCJR-SISO decoders, this parallelism leads to an on-chip communication issue related to the interleaver. Indeed, interleaving has to be parallelized in order to extend proportionally the communication bandwidth. In consequence, the complexity of communication structure (and also communication time) increases with parallelism degree and access conflicts may occur (even if latest standards use conflict-free interleavers). The latter problem can be resolved using an interleaving mapping to avoid conflicts [8] or using a communication structure to manage conflicts on the fly [9].

3.2.2. Component-Decoder Parallelism. The component-decoder parallelism is a new kind of parallelism that has become operational with the introduction of the shuffled decoding technique [10]. The basic idea of the shuffled decoding technique is to execute all component decoders in parallel and to exchange extrinsic information as soon as created. Using this method, the iteration period is halved in comparison with originally proposed serial turbo decoding [10]. Nevertheless, component-decoder parallelism may require additional iterations as explained in Section 5.

This level of parallelism can reach a reasonable parallelism degree and preserve memory area. Due to its great potential for scalability and mastered area overhead, new explorations are focused on this second level of parallelism.

3.2.3. Turbo-Decoder Level Parallelism. The highest level of parallelism duplicates the whole turbo decoder to process frames in parallel. Iteration parallelism occurs in a pipelined fashion (each parallel instance works on different frames and transmits their results to the instance in charge of the next iteration) with a maximum pipeline depth equal to twice the iteration number, whereas frame parallelism (each parallel instance decodes completely its frames) presents no limitation in parallelism degree. Nevertheless, turbo-decoder level parallelism is too area-expensive (all memories and computation resources are duplicated) and presents no gain in decoding latency and thus it is not considered in this work.

4. Initialization in Subblock Parallelism

As described in Section 3, subblock parallelism takes place at frame level and requires initializations. Proper initialization is mandatory to achieve correct decoding since information on recursion metrics is available at frame ending points, but not at subblock ending points. Estimation of undetermined information can be obtained either by acquisition or by message passing between neighboring subblocks.

4.1. Initialization by Acquisition. This widely used initialization method consists in estimating recursion metrics by means of an overlapping region called acquisition window or prolog. Starting from a trellis section, where all the states are initialized to a uniform constant, the acquisition window will be processed on its length, denoted AL, to provide reliable recursion metrics at subblock ending points. This acquisition length is determined at design time in order to achieve negligible error-rate degradation. It is fixed according to the number of redundancy bits in the prolog (typically 6 bits). Another empirical rule recommends from 3 to 5 times the constraint length of the code for this acquisition length [17].

To evaluate this initialization method, we assume that the butterfly scheme is used in each subblock and that information is available concerning the beginning and ending states of the frame (circular trellis or tail-biting). So, the decoding computation time of an architecture with d_{SB} subblocks (i.e., the parallelism degree is d_{SB}) initialized with acquisition method is:

$$\begin{gathered} t_{\mathrm{SB}}(d_{\mathrm{SB}}) = \mathrm{dec}\cdot \mathrm{it}\left(\left(\frac{N}{d_{\mathrm{SB}}} + \mathrm{AL}\right)t_{\mathrm{sym}} + t_p(d_{\mathrm{SB}})\right) \quad \text{for } d_{\mathrm{SB}} > 1, \\ t_{\mathrm{SB}}(1) = \mathrm{dec}\cdot \mathrm{it}\left(N t_{\mathrm{sym}} + t_p(1)\right), \end{gathered} \tag{5}$$

where dec represents the number of component decoders (typically 2), it the number of iterations (note that it is independent of d_{SB}), N the frame length in symbols, t_{sym} the time required to process one symbol and $t_p(d_{\mathrm{SB}})$ the propagation time spent by extrinsic information in communication resources between component decoders (note that t_p is usually negligible). The corresponding speedup is:

$$\begin{aligned} S_{\mathrm{SB}}(d_{\mathrm{SB}}) &= \frac{t_{\mathrm{SB}}(1)}{t_{\mathrm{SB}}(d_{\mathrm{SB}})} = \frac{N t_{\mathrm{sym}} + t_p(1)}{(N/d_{\mathrm{SB}} + \mathrm{AL})t_{\mathrm{sym}} + t_p(d_{\mathrm{SB}})} \\ &= \frac{1 + t_p(1)/N t_{\mathrm{sym}}}{1/d_{\mathrm{SB}} + \mathrm{AL}/N + t_p(d_{\mathrm{SB}})/N t_{\mathrm{sym}}}. \end{aligned} \tag{6}$$

This speedup follows an Amdahl's law where the sequential and incompressible part is mainly related to computations of acquisition windows, which are mandatory to reliable initializations. Thus subblock parallelism with initialization by acquisition encounters a throughput ceiling value and the maximum speedup (obtained when d_{SB} equals N) is roughly equal to $N/(\mathrm{AL}+1)$ (by neglecting t_p).

4.2. Initialization by Message Passing. The second method initializes dynamically a subblock with recursion metrics computed during the last iteration in the neighboring subblocks [16]. So this technique does not require additional memory except for on-chip communication resources between BCJR SISO units. It was shown in [7] that the asymptotic error-rate, that is, the error-rate achieved with an infinite number of iterations, is not affected by the message passing approach whatever the parallelism degree. Consequently, it ensures that initialization by message passing can be used without error-correction performance degradation at the expense of additional iterations.

Let $\mathrm{it}(d_{\mathrm{SB}})$ be the mean number of iterations for an architecture with d_{SB} subblocks communicating with the message passing technique. For accuracy reasons, $\mathrm{it}(d_{\mathrm{SB}})$ is obtained with the genie stopping criterion, that is, the

Table 1: Message passing parameter for different code rates for double binary codes (DVB-RCS).

Code rate	6/7	3/4	1/2	1/3
f	21.5	13.5	8	6.5

decoding is stopped immediately after an iteration returning the right codeword and is not started for undecodable frame.

Through large number of simulations, we observe that $\mathrm{it}(d_{\mathrm{SB}})$ increases linearly with the parallelism degree and its slope varies with the inverse of the block size. Thus, we can express empirically it as:

$$\mathrm{it}(d_{\mathrm{SB}}) = \mathrm{it}(1)\left(1 + \frac{f}{N}(d_{\mathrm{SB}} - 1)\right), \tag{7}$$

where f is a constant value, that depends of the code rate and components codes.

For example, Table 1 gives the constant value f for different code rates of double binary codes. Note that f increases with the code rate since a lower code rate makes it possible to provide reliable metrics (recursion and extrinsic information) on shorter subblock size and, consequently, are less affected by parallelism effects.

Thus, the decoding time with this initialization technique can be refined in

$$\begin{aligned} t_{\mathrm{SB}}(d_{\mathrm{SB}}) &= \mathrm{dec}\cdot \mathrm{it}(d_{\mathrm{SB}})\left(\frac{N}{d_{\mathrm{SB}}}t_{\mathrm{sym}} + t_p(d_{\mathrm{SB}})\right) \\ &= \mathrm{dec}\cdot \mathrm{it}(1)\left(1 + \frac{f}{N}(d_{\mathrm{SB}} - 1)\right)\left(\frac{N}{d_{\mathrm{SB}}}t_{\mathrm{sym}} + t_p(d_{\mathrm{SB}})\right), \end{aligned} \tag{8}$$

and its speedup is

$$\begin{aligned} S_{\mathrm{SB}}(d_{\mathrm{SB}}) &= \frac{t_{\mathrm{SB}}(1)}{t_{\mathrm{SB}}(d_{\mathrm{SB}})} \\ &= \frac{\mathrm{it}(1)\left(N\ t_{\mathrm{sym}}+t_p(1)\right)}{\mathrm{it}(1)(1+(f/N)(d_{\mathrm{SB}}-1))\left((N/d_{\mathrm{SB}})t_{\mathrm{sym}}+t_p(d_{\mathrm{SB}})\right)} \\ &= \frac{1+t_p(1)/Nt_{\mathrm{sym}}}{((1-f/N)/d_{\mathrm{SB}}+f/N)\left(1+d_{\mathrm{SB}}t_p(d_{\mathrm{SB}})/Nt_{\mathrm{sym}}\right)}. \end{aligned} \tag{9}$$

Like subblock parallelism with initialization by acquisition, an Amdahl's law can be recognized in this speedup equation if t_p terms are neglected. Thus, the sequential part of the subblock parallelism with message passing initialization is around f/N (compared to AL/N for acquisition initialization). Similarly, we can show (by neglecting t_p terms and by remarking that f is much smaller than N) that the maximum speedup (obtained when d_{SB} equals N) is roughly equal to $N/(f+1)$.

4.3. Subblock Parallelism Efficiency and Performance Comparison. To evaluate the efficiency of initialization methods, a comparison of their subblock parallelism speedups is accurate enough since the ratios (r_{dup}) between the BCJR-SISO decoder area and the overall architecture area are very close for both methods (the initialization methods only require memorization overheads, that are negligible with respect to the BCJR-SISO decoder area).

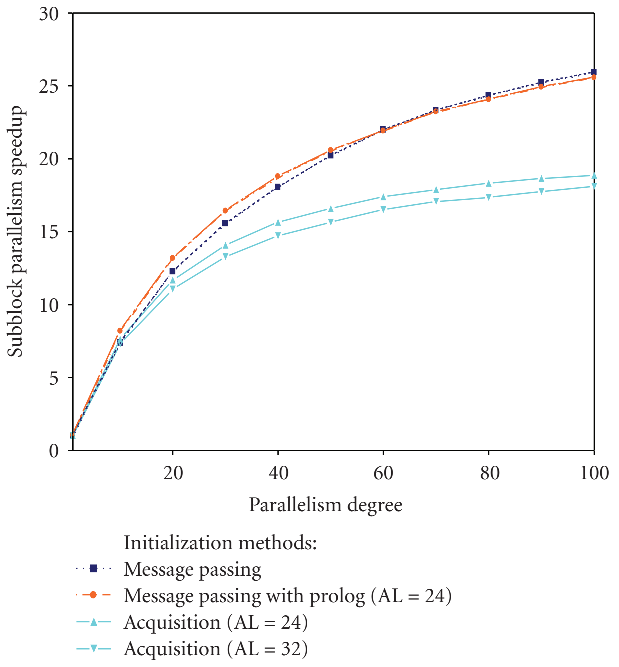

Figure 2: Simulated normalized speedup for several initialization methods, DVB-RCS, $R = 6/7$, 188 bytes frame, $E_b/N_0 = 4.2$ dB, 5 bit quantization, Max-Log-MAP algorithm.

Each subblock parallelism speedup takes into account acquisition overhead and additional iterations. Note that using additional iterations with the acquisition method has quite no effect on error-correction performance.

For example, Figure 2 compares subblock parallelism normalized speedups obtained by simulations of a double binary turbo code at rate 6/7 for several initialization methods: message passing, acquisition with 24 and 32 symbols and message passing with a 24-symbol prolog (i.e., an acquisition is just perform for the first iteration). The figure shows clearly that the message passing technique outperforms the acquisition one whatever the acquisition length. The difference becomes sharper at high subblock parallelism degree. These results were also validated by simulations under various conditions (code rate, block size). Concerning the initialization of the first iteration, a prolog provides more reliable initialization than a uniform initialization. This enables to reduce the iteration number. But this reduction only improves the speedup if the gain is larger than the prolog overhead. In fact, improvements with prolog are only obtained for high code rate and at low parallelism degrees.

Concerning error-rate performance, initialization by message passing induces no degradation as stated in the parallelism efficiency definition, while initialization by acquisition may degrade error-rate performance, if the acquisition length is not long enough. For example, Figure 3 illustrates

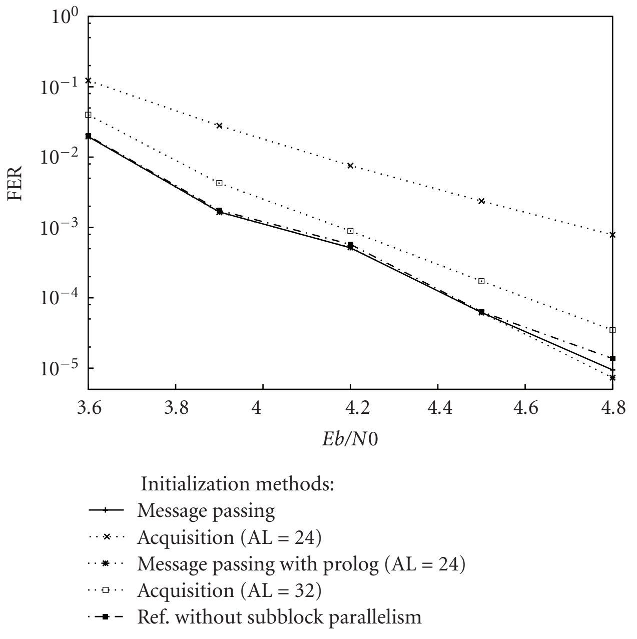

Figure 3: FER and initialization methods for high parallelism degree (47), DVB-RCS, $R = 6/7$, 188 bytes frame, 5 bit quantization, Max-log-MAP algorithm.

Frame Error Rate (FER) performance for the same initialization methods and conditions as Figure 2 considering a DVB-RCS code [18] with a 47 subblock parallelism degree. In this case, the initialization with 32-symbol acquisition length implies a 0.15 dB degradation in comparison with the message passing initialization or the reference curve without parallelism.

Comparison between both techniques tends clearly in favor of the message passing technique, which enables better error-rate performance and better subblock parallelism efficiency. However, whatever the initialization method, the subblock parallelism speedup reaches a ceiling at high parallelism degrees. Therefore, in order to find the parallelism degree maximizing the architecture efficiency, we have to consider the subblock parallelism efficiency. For the message passing method, the subblock parallelism efficiency is:

$$E_{SB}(d_{SB}) = \frac{S_{SB}(d_{SB})}{O(d_{SB})} = \frac{1+\varepsilon}{\left(1 + r_{dup}(d_{SB}-1)\right)\left((1-f/N)/d_{SB} + f/N\right)}, \tag{10}$$

where $1+\varepsilon = (1 + t_p(1)/Nt_{sym})/(1 + d_{SB}t_p(d_{SB})/Nt_{sym})$.

The first derivative of the efficiency (once neglecting the t_p terms) equals zero when its numerator does

$$\frac{\left(1-r_{dup}\right)(1-f/N)}{d_{SB}^2} - r_{dup}\frac{f}{N} = 0. \tag{11}$$

Thus, the subblock parallelism efficiency has an extremum when the parallelism degree is equal to $\sqrt{(1/r_{dup}-1)(N/f-1)}$. Indeed, this extremum is a maximum as the first derivative is positive below this value (the denominator is always positive) and negative above. The maximum efficiency depends on r_{dup}, on the block size, code rate and component decoders. Above this value, the architecture efficiency is degraded and other parallelisms have to be considered such as component-decoder parallelism.

5. Component-Decoder Parallelism Analysis

As described in Section 3.2.2, component-decoder parallelism takes advantage of the shuffled decoding technique that executes all component decoders in parallel and exchanges extrinsic information as soon as created. However, the relevance of this technique depends greatly on other parallelism choices and on interleaving rules.

5.1. Shuffled Decoding Speedup.

5.1. Shuffled Decoding Speedup. By definition, the shuffled decoding process starts a new iteration without waiting the propagation of extrinsic information through the communication resources. In other words, decoding and information exchange are performed concurrently in shuffled decoding. Thus, the shuffled decoding time only depends on the time needed to process a subblock and can simply be expressed as:

$$t_{shuffled}(d_{shuffled}, d_{SB}) = \text{it}_{shuffled}(d_{shuffled}, d_{SB})\frac{N}{d_{SB}}t_{sym}, \tag{12}$$

where $d_{shuffled}$ is the component-decoder parallelism degree (equals to dec for shuffled decoding), d_{SB} the subblock parallelism degree used in each component decoder, and $\text{it}_{shuffled}$ the number of iterations required by the shuffled decoding process.

Then, by dividing the decoding time of the serial process with a subblock parallelism degree of d_{SB}—see (8)—by the shuffled decoding time, we get the shuffled decoding speedup

$$\begin{aligned} S_{shuffled}&(d_{shuffled}, d_{SB}) \\ &= \frac{t_{serial}(d_{SB})}{t_{shuffled}(d_{shuffled}, d_{SB})} \\ &= \frac{\text{dec}\cdot\text{it}_{serial}(d_{SB})\left((N/d_{SB})t_{sym} + t_p(d_{SB})\right)}{\text{it}_{shuffled}(d_{shuffled}, d_{SB})(N/d_{SB})t_{sym}} \\ &= d_{shuffled}\frac{\text{it}_{serial}(d_{SB})}{\text{it}_{shuffled}(d_{shuffled}, d_{SB})}\left(1 + \frac{d_{SB}t_p(d_{SB})}{Nt_{sym}}\right). \end{aligned} \tag{13}$$

The impact of the propagation time will be discussed in Section 5.3. In Section 5.2, a zero propagation time will be considered and the convergence speed of the shuffled decoding process, defined as $CS_{shuffled} = \text{it}_{serial}(d_{SB})/\text{it}_{shuffled}(d_{shuffled}, d_{SB})$, is analysed.

Simulation results demonstrate that the convergence speed of shuffled decoding ranges from 0.6 to 0.95 depending on the choice of parallelism techniques and on the interleaving scheme.

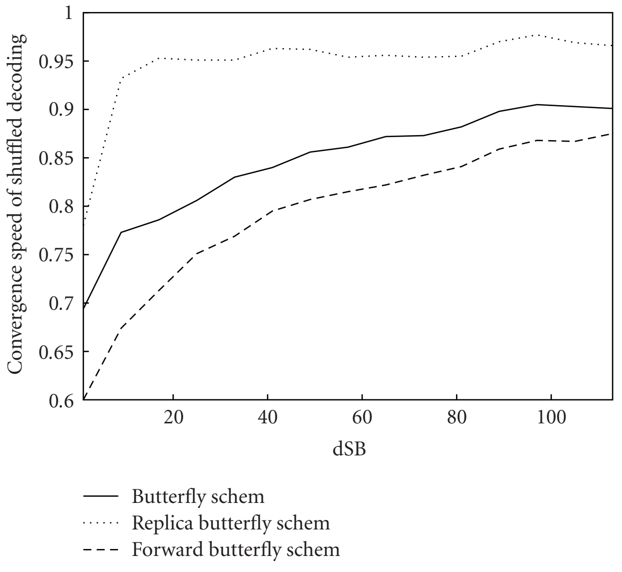

Figure 4: Evolution of convergence speed of shuffled decoding with subblock parallelism degree, message passing initialization, random interleaver, $R = 6/7$, 188 byte frame.

5.2. Combining Shuffled Decoding with Other Parallelism Techniques. Considering to BCJR computation parallelism and subblock parallelism, the convergence speed of shuffled decoding is very disparate.

In [15], the influence of BCJR computation scheme was pointed out with the replica scheme proposal. The replica principle consists in generating two extrinsic information per symbol per iteration (one in the forward and one in the backward), instead of one extrinsic information in other schemes. Thus, decoders have more up-to-date extrinsic information and converge faster. Using the schemes presented in Figure 1, our simulations also reveal that shuffled decoding convergence speed is always the greatest with the replica butterfly scheme (convergence speed observed around 0.8) and the lowest with the forward butterfly scheme (convergence speed observed around 0.6). However, in these conditions, shuffled decoding is not valuable for implementation, since its resulting speedup is less than the one obtained using subblock parallelism (same number of BCJR-SISO decoders) instead of component decoder parallelism.

Therefore, we proposed in [7] to combine component-decoder parallelism and subblock parallelism in order to improve the convergence speed of shuffled decoding. By definition, this latter is the iteration number required by a sequential execution of component decoders (using d_{SB} BCJR-SISO decoders) over the iteration number required with shuffled decoding (i.e., d_{shuffled} d_{SB} BCJR-SISO decoders working in parallel). Figure 4 illustrates the behavior of shuffled decoding convergence speed with respect to subblock parallelism degree. Whatever the BCJR computation scheme, we can observe that convergence speed increases with the subblock parallelism degree. Additionally, the convergence speed with butterfly and forward butterfly schemes tends, at high subblock parallelism, towards the convergence speed with replica scheme. Note that this latter for high subblock parallelism degree is close to the perfect convergence speed (equal to 1). In these conditions, component-decoder parallelism has to be used rather than subblock parallelism, which suffers from Amdahl's law.

To help the selection of parallelism techniques and schemes for hardware implementation, the parallelism efficiency of BCJR-SISO decoder level parallelism (i.e., including subblock and component-decoder parallelisms) has to be considered

$$\begin{aligned} E(d_{\text{shuffled}}, d_{\text{SB}}) &= \frac{t_{\text{serial}}(1)/t_{\text{shuffled}}(d_{\text{shuffled}}, d_{\text{SB}})}{O(d_{\text{shuffled}} d_{\text{SB}})} \\ &= \frac{S_{\text{SB}}(d_{\text{SB}}) S_{\text{shuffled}}(d_{\text{shuffled}}, d_{\text{SB}})}{1 + r_{\text{dup}}(d_{\text{SB}} d_{\text{shuffled}} - 1)}, \end{aligned} \tag{14}$$

where S_{SB} (resp. S_{shuffled}) is the subblock parallelism speedup (resp. the shuffled decoding speedup). Note that the area overhead is considered to be similar for both parallelisms. Consequently, the global overhead depends on the product of the parallelism degrees.

Using the subblock parallelism speedup obtained with the message passing technique (9), the parallelism efficiency can be refined in

$$\begin{aligned} &E(d_{\text{shuffled}}, d_{\text{SB}}) \\ &\quad = \frac{d_{\text{SB}} d_{\text{shuffled}} C S_{\text{shuffled}} \left(1 + t_p(1)/N t_{\text{sym}}\right)}{\left(1 - f/N + (f/N) d_{\text{SB}}\right)\left(1 + r_{\text{dup}}(d_{\text{SB}} d_{\text{shuffled}} - 1)\right)}. \end{aligned} \tag{15}$$

As the convergence speed varies nonlinearly with d_{SB}, the parallelism degree that maximizes the parallelism efficiency can not be expressed like in Section 4. Therefore, the parallelism efficiency is represented in Figure 5 in a typical implementation context example for four configurations initializing subblocks with message passing technique. The first configuration only uses subblock parallelism and offers the best result for low parallelism degree. The other configurations use additionally component-decoder parallelism with the butterfly scheme, the forward butterfly scheme or the replica butterfly scheme. We can observe that configurations using shuffled decoding enable better speedup for high parallelism degree. In this typical implementation context, a BCJR-SISO decoder represents 25% of the circuit area with butterfly and replica butterfly schemes and 18% of the circuit area with forward butterfly scheme. Due to its lower complexity, the configuration with the forward butterfly scheme is the most efficient for high parallelism degree. However, it is more interesting to find the configuration maximizing the efficiency of BCJR-SISO decoder level parallelism and, in consequence, the architecture efficiency. Indeed, from this maximal point, it becomes more efficient to increase the throughput using turbo-decoder level parallelism (without effect on architecture efficiency) than using BCJR-SISO decoder level parallelism. In the presented case, this point is obtained with the configuration using shuffled decoding and replica butterfly scheme.

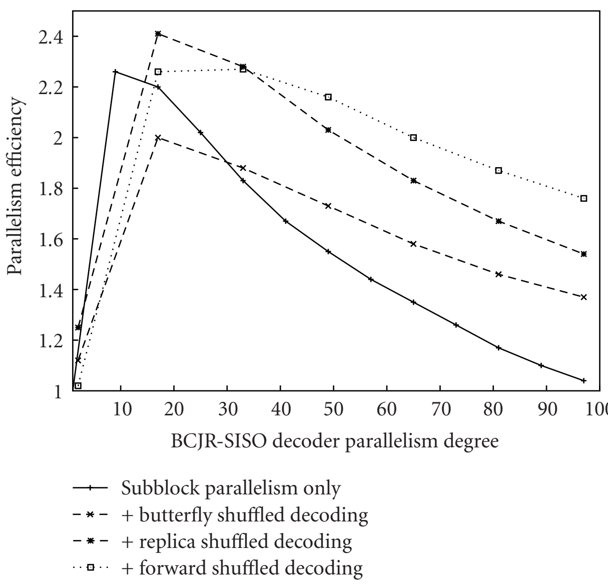

Figure 5: Parallelism efficiency with respect to BCJR-SISO decoder parallelism degree, message passing initializations, random interleaver, $R = 6/7$, 188 byte frame, $t_p = 0$, $r_{\text{dup}} = 25\%$ (butterfly, replica) or $= 18\%$ (forward).

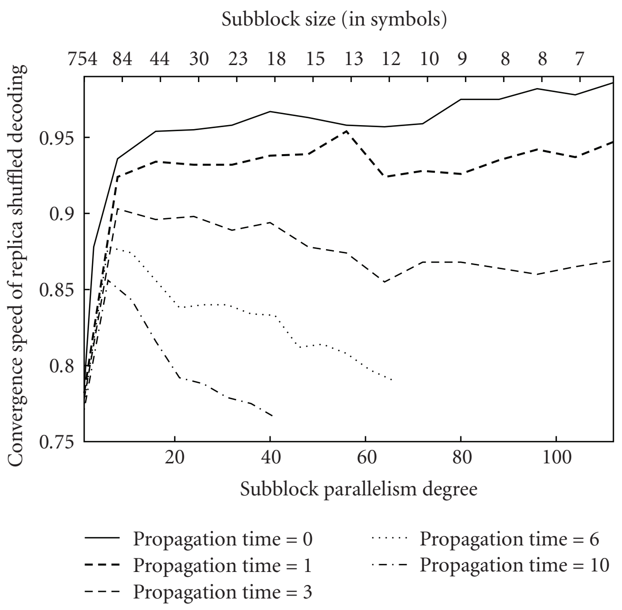

Figure 6: Convergence speed of replica shuffled decoding and propagation time, random interleaver, $R = 6/7$, 188 byte frame.

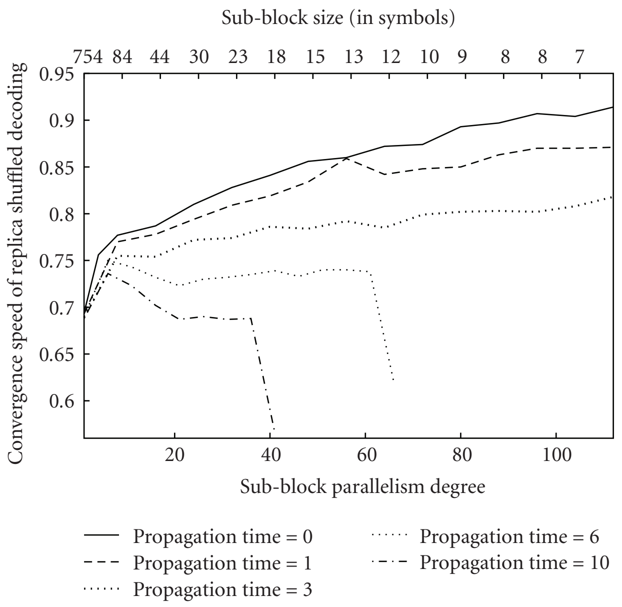

Figure 7: Convergence speed of butterfly shuffled decoding and propagation time, random interleaver, $R = 6/7$, 188 byte frame.

5.3. Shuffled Decoding Implementation Issues

5.3.1. Propagation Time Effect on Shuffled Decoding. In real implementations of a parallel turbo decoder, exchanged extrinsic information is not immediately updated in the targeted SISO. We define the propagation time as the time required to update the extrinsic information value of a symbol. It includes the computation time of the extrinsic information value and the communication time. As complex communication structures (complexity increasing with subblock parallelism) are needed to perform interleaving without conflicts, communication time is particularly not negligible. However, for most of hardware implementations, propagation time is less than $3t_{\text{sym}}$.

Because of this propagation time, consistency conflicts (i.e., a component decoder performs a read access before the write access of the other component decoder is completed.) may occur in extrinsic information memory. Hence, the symbols suffering from consistency conflict must have one memory bank per decoder and their extrinsic information values are exchanged in the time interval of two iterations instead of one for other symbols. Consequently, the convergence of the shuffled decoding process is slowed down.

With simulations using random interleavers, we observe the mean degradation on the convergence speed implied by the propagation time for replica shuffled decoding (Figure 6) and for butterfly shuffled decoding (Figure 7). Time unit is the time required to process one symbol (t_{sym}). In both figures, the convergence speed slows down when propagation time increases, but replica scheme is more robust. In addition, we can observe that degradations become significant when processing time of a subblock becomes comparable with propagation time. In this case, the percentage of symbol reliabilities not updated in the running iteration becomes very high. For example, in Figure 7, the limit where no symbol reliabilities are updated during the iteration is observed for propagation times 6 and 10. In this case, convergence of the butterfly shuffled decoding process tends towards 0.5. This means that the shuffled process spends the same computation time as the serial process to converge. Then, the speedup is only obtained through communication time, which is concurrent to computation time for shuffled decoding.

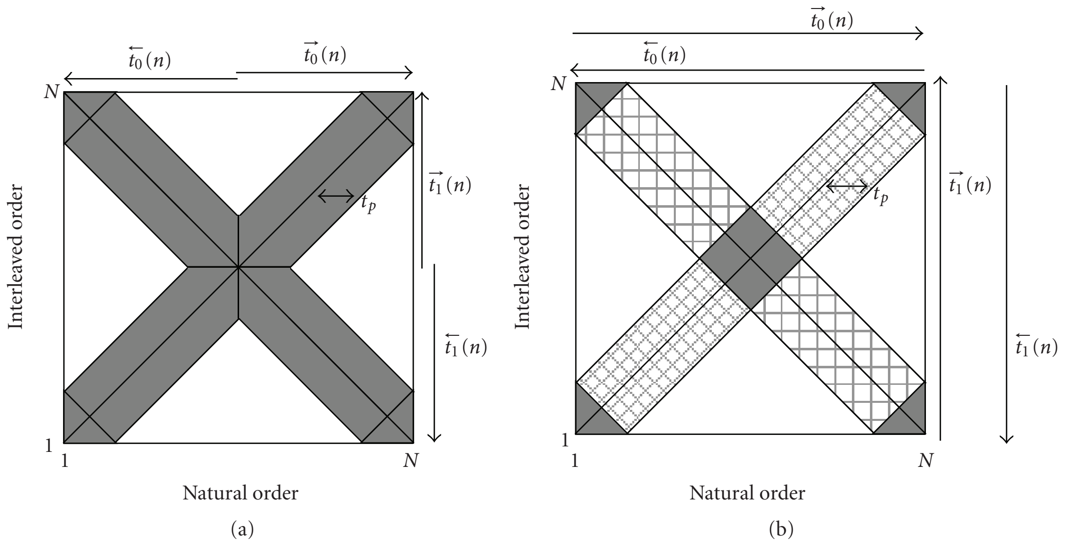

FIGURE 8: Interleaver masks for: (a) butterfly scheme and (b) replica butterfly scheme.

Thus, preserving a good convergence speed of the shuffled process requires the preservation of a reasonable percentage of exchanging symbol reliabilities in the process. To better understand the influence of propagation time and interleaving rules on this percentage, we propose a geometrical explanation.

5.3.2. Geometrical Explanation of Propagation Time Effect. Let S_j^k be the SISO decoder that processes the kth subblock of the frame on the jth component decoder (e.g., $j = 0, 1$ for a concatenation of two convolutional codes). Let t_p denote the propagation time. According to the considered scheme of BCJR computations (Section 3.1.2), at the ith iteration, S_j^k can deliver extrinsic information for the estimated symbol $\hat{u}_n$ in the forward recursion or/and in the backward recursion. Extrinsic information transmission time will be denoted $\overrightarrow{t_j}(n)$ for transmission in forward direction and $\overleftarrow{t_j}(n)$ for transmission in backward direction.

To avoid propagation time effect for a symbol, updating extrinsic information has to happen before processing for each SISO decoder. As forward and backward transmissions can be combined in both component decoders, each symbol has four possibilities to be updated during one iteration and one is sufficient for a good convergence. Thus, considering the interleaver Π, the previous condition is expressed by:

$$\begin{gathered}\left|\overrightarrow{t_0}(n) - \overrightarrow{t_1}(\Pi(n))\right| > t_p\\ \text{or}\\ \left|\overrightarrow{t_0}(n) - \overleftarrow{t_1}(\Pi(n))\right| > t_p\\ \text{or}\\ \left|\overleftarrow{t_0}(n) - \overrightarrow{t_1}(\Pi(n))\right| > t_p\\ \text{or}\\ \left|\overleftarrow{t_0}(n) - \overleftarrow{t_1}(\Pi(n))\right| > t_p.\end{gathered} \tag{16}$$

To describe the interleaving design rule, the two-dimensional representation of an interleaver introduced in [19] is well suited. In this representation, the natural order (resp. interleaved) is depicted on the horizontal-axis (resp. vertical-axis) and the symbol with index n in natural order is represented with the point $(n, \Pi(n))$. On this representation, (16) is translated into banned regions constituting the *mask* of the interleaver [20]. So, symbols have a slower convergence inside the interleaver mask than outside.

Depending on BCJR computation schemes, (16) can define various interleaver masks. For example, Figure 8 represents interleaver masks of butterfly and replica butterfly schemes. The first example (a), the butterfly scheme is used by both component decoders. For this scheme, $\overrightarrow{t_j}$ and $\overleftarrow{t_j}$ are defined only on the second half of processing time and have exclusive existence for each symbol. Consequently, each symbol is only concerned by one inequality and the four inequalities define four half-diagonal banned regions. Combining these banned regions constitutes the interleaver mask associated to butterfly scheme. Note that the mask covers the entire space as soon as the propagation time is equal to half the frame length. In these conditions, extrinsic information exchanges need exactly the time of two iterations. This means that the convergence speed of shuffled decoding in this case is one half. This result exactly match with the simulation results previously observed in Figure 7.

For the interleaver mask of replica butterfly scheme (Figure 8(b)), $\overrightarrow{t_j}$ and $\overleftarrow{t_j}$ are both defined for each symbol. Hence, each symbol is concerned by the four inequalities, which define four diagonal banned regions. Thus, the interleaver mask is the intersection of these diagonals, that is, the square in the middle and the one distributed on the corners (iterations are assumed to be executed continuously, that is, the first and last symbols of the frame are neighbors in processing time). In comparison with butterfly mask, the replica mask is smaller. So, in random interleaving case, the replica mask allows more exchanges and, consequently, enables a better convergence of the iterative process. Like for butterfly mask, the replica mask covers the entire space when t_p is equal to $N/2$. However, convergence speed is not slowed down to 0.5 (see Figure 7) since, with replica scheme, additional extrinsic information exchanges exist in the time interval of one iteration. Indeed, one information update is used at two different instants (forward and backward), but (16) only consider the closest instant. Additional exchanges are the secondary reason of the robustness of replica shuffled decoding to long propagation time.

5.3.3. Concluding Remark on Real Shuffled Decoding. Finally, in a real implementation context (propagation time less than 3 t_{sym}), the propagation time effect on the convergence speed (and efficiency) of the shuffled decoding process introduces loss always less than 10% for replica shuffled decoding and less than 12% for butterfly shuffled decoding. Therefore, with the efficiency gap observed in Figure 5 between implementations using only subblock parallelism and implementations using ideal shuffled decoding, we can conclude that real shuffled decoding implementations based on replica and butterfly schemes are more efficient for high parallelism degree.

6. Conclusion

This paper thoroughly examines parallel convolutional turbo decoding. We have analyzed and classified the various parallelism techniques that could be used in convolutional turbo decoding with the BCJR algorithm. The three-level classification proposed includes: *BCJR metric level parallelism, BCJR SISO decoder level parallelism,* and *Turbo-decoder level parallelism.* Considering its achievable high parallelism degrees and mastered area overhead, our analyses focus on the BCJR SISO decoder level, which includes subblock parallelism and component-decoder parallelism. On the one hand, we demonstrate that subblock initialization is more efficient with the message passing technique than with the acquisition technique and also that subblock parallelism becomes inefficient for high subblock parallelism degrees. On the other hand, we show that component-decoder parallelism efficiency depends on the BCJR computation scheme, on subblock parallelism degree, and on propagation time. Furthermore, results point out that the shuffled decoding, until now never considered in hardware implementation, improves the efficiency of very high-throughput low-latency implementations.

Acknowledgment

This work was supported in part by the UDEC project of the French Research Agency (ANR).

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near SHANNON limit error-correcting coding and encoding: turbo-codes," in *Proceedings of the IEEE International Conference on Communications (ICC '93)*, pp. 1064–1070, Geneva, Switzerland, May 1993.

[2] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, 1974.

[3] 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, 2003.

[4] G. Masera, G. Piccinini, M. R. Roch, and M. Zamboni, "VLSI architectures for turbo codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 3, pp. 369–379, 1999.

[5] M. M. Mansour and N. R. Shanbhag, "VLSI architectures for SISO-APP decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 4, pp. 627–650, 2003.

[6] C. Schurgers, F. Catthoor, and M. Engels, "Memory optimization of MAP turbo decoder algorithms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 2, pp. 305–312, 2001.

[7] O. Muller, A. Baghdadi, and M. Jezequel, "Exploring parallel processing levels for convolutional turbo decoding," in *Proceedings of International Conference on Information & Communication Technologies: From Theory to Applications (ICTTA '06)*, April 2006.

[8] A. Tarable, S. Benedetto, and G. Montorsi, "Mapping interleaving laws to parallel turbo and LDPC decoder architectures," *IEEE Transactions on Information Theory*, vol. 50, no. 9, pp. 2002–2009, 2004.

[9] M. J. Thul, F. Gilbert, T. Vogt, G. Kreiselmaier, and N. Wehn, "A scalable system architecture for high-throughput turbo-decoders," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 39, no. 1-2, pp. 63–77, 2005.

[10] J. Zhang and M. P. C. Fossorier, "Shuffled iterative decoding," *IEEE Transactions on Communications*, vol. 53, no. 2, pp. 209–213, 2005.

[11] O. Muller, A. Baghdadi, and M. Jézéquel, "On the parallelism of convolutional turbo decoding and interleaving interference," in *Proceedings of the Global Telecommunications Conference (GLOBECOM '06)*, December 2006.

[12] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *AFIPS Spring Joint Computing Conference*, pp. 483–485, April 1967.

[13] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and suboptimal maximum a posteriori algorithms suitable for turbo decoding," *European Transactions on Telecommunications*, vol. 8, no. 2, pp. 119–125, 1997.

[14] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Soft-input soft-output maximum a posteriori (MAP) module to decode parallel and serial concatenated codes," TDA Progress Report, 1996.

[15] J. Zhang, Y. Wang, M. Fossorier, and J. S. Yedidia, "Replica shuffled iterative decoding," in *Proceedings of the IEEE International Symposium on Information Theory*, Adelaide, Australia, September 2005.

[16] S. Yoon and Y. Bar-Ness, "A parallel MAP algorithm for low latency turbo decoding," *IEEE Communications Letters*, vol. 6, no. 7, pp. 288–290, 2002.

[17] T. Wolf, "Initialization of sliding windows in turbo decoders," in *Proceedings of the 3rd International Symposium on Turbo Codes and Related Topics*, pp. 219–222, Brest, France, September 2003.

[18] C. Douillard, M. Jezequel, C. Berrou, N. Brengarth, J. Tousch, and N. Pham, "The turbo code standard for DVB-RCS," in

Proceedings of the 2nd International Symposium on Turbo Codes & Related Topics, pp. 535–538, Brest, France, 2000.

[19] C. Heegard and S. B. Wicker, *Turbo Coding*, Kluwer Academic Publishers, Dodrecht, The Netherlands, 1999.

[20] D. Gnaëdig, E. Boutillon, J. Tousch, and M. Jezequel, "Towards an optimal parallel decoding of turbo codes," in *Proceedings of the 4th International Symposium on Turbo Codes and Related Topics*, Munich, Germany, April 2006.