Circuits Syst Signal Process (2016) 35:2832–2854
DOI 10.1007/s00034-015-0168-4

Memory-Reduced Maximum A Posteriori Probability Decoding for High-Throughput Parallel Turbo Decoders

Rahul Shrestha[1] · **Roy Paily**[2]

Received: 30 May 2014 / Revised: 8 September 2015 / Accepted: 9 September 2015 /
Published online: 22 September 2015

Abstract Wireless communication standards make use of parallel turbo decoder for higher data rate at the cost of large hardware resources. This paper presents a memory-reduced back-trace technique, which is based on a new method of estimating backward-recursion factors, for the maximum a posteriori probability (MAP) decoding. Mathematical reformulations of branch-metric equations are performed to reduce the memory requirement of branch metrics for each trellis stage. Subsequently, an architecture of MAP decoder and its scheduling based on the proposed back trace as well as branch-metric reformulation are presented in this work. Comparative analysis of bit-error-rate (BER) performances in additive white Gaussian noise channel environment for MAP as well as parallel turbo decoders are carried out. It has shown that a MAP decoder with a code rate of 1/2 and a parallel turbo decoder with a code rate of 1/3 have achieved coding gains of 1.28 dB at a BER of 10^{-5} and of 0.4 dB at a BER of 10^{-4}, respectively. In order to meet high-data-rate benchmarks of recently deployed wireless communication standards, very large scale integration implementations of parallel turbo decoder with 8–64 MAP decoders have been reported. Thereby, savings of hardware resources by such parallel turbo decoders based on the suggested memory-reduced techniques are accounted in terms of complementary metal oxide semiconductor transistor count. It has shown that the parallel turbo decoder with 32 and 64 MAP decoders has shown hardware savings of 34 and 44 % respectively.

✉ Rahul Shrestha
rahul.shrestha@iiit.ac.in

Roy Paily
roypaily@iitg.ernet.in

[1] Center for VLSI and Embedded System Technologies, IIIT-Hyderabad, Gachibowli, Hyderabad, Telangana 500032, India

[2] Department of Electronics and Electrical Engineering, IIT-Guwahati, Guwahati, Assam 781039, India

Keywords Sliding-window Bahl–Cocke–Jelinek–Raviv (SWBCJR) algorithm · Maximum a posteriori probability (MAP) decoder · Turbo code · Parallel turbo decoder · Bit-error-rate (BER) performance · Very large scale integration (VLSI) design

1 Introduction

In this era of multimedia communication, the latest wireless standards target higher data rate to meet the requirement of various service demands of customers. Turbo code ensures reliable communication in many of these recent standards and is used by forward error correction unit in their physical layer [2]. Near-optimal error-rate performance delivered by an iterative process of turbo code is the reason for its supremacy over other channel coding techniques [26,27] and makes it acceptable by wireless communication standards such as digital video broadcasting–satellite services to handhelds (DVB-SH), high-speed downlink packet access (HSDPA), IEEE 802.16e, IEEE 802.16m, third-generation partnership project long-term evolution (3GPP-LTE) and evolving LTE-Advanced [19]. Integral parts of turbo decoder are maximum a posteriori probability (MAP) decoder and pseudorandom interleaver [20,30]. However, the conventional turbo decoder with non-parallel architecture is unable to achieve higher data rates over 300 Mbps and 1 Gbps of 3G (such as 3GPP-LTE) and 4G (such as LTE-Advanced) wireless communication standards, respectively [3,6]. On the other hand, turbo decoder with parallel architecture of multiple MAP decoders can achieve such data rates [7], and various contributions have been reported for the design of such decoders [12,28,36–38]. Figure 1 shows a conventional turbo decoder with parallel architecture for higher data-rate application. Soft-demodulated input soft values of received bits, at the receiver side of communication system, are stored in the stack of memories (MEM) at the decoder input. As shown in Fig. 1, outputs of these memories are linked with multiple MAP decoders via inter-connecting networks (ICNWs) which route input soft values from memories, either sequentially or pseudorandomly based on the interleaved addresses, to their respective MAP decoders. After processing the input soft values, the extrinsic information produced by these decoders is stored in MEM. Finally, the extrinsic information outputs from these memories are fed back to MAP decoders via ICNW and they are used as a priori probabilities in the iterative process of decoding, as shown in Fig. 1. Though the parallel turbo decoder can achieve higher data rate, it demands huge amount of hardware resources. Reduction in hardware requirement in parallel turbo decoder has been a motivation for our work presented in this paper. Specifically, an objective of this work is to improve the hardware savings in parallel turbo decoders by reducing the memory required for storing forward-recursion factors and branch metrics in each MAP decoders used in such parallel architectures. Some of the works with similar motivation have been reported in [13,17,24,31,33,34]. A memory reduction technique based on metric compression using non-uniform quantization and Walsh-Hadamard transform is presented in [24]. Another approach [17] is based on low-power traceback of MAP-based duo-binary turbo decoders. Reduction in branch-metric memory and scheduling the back trace of

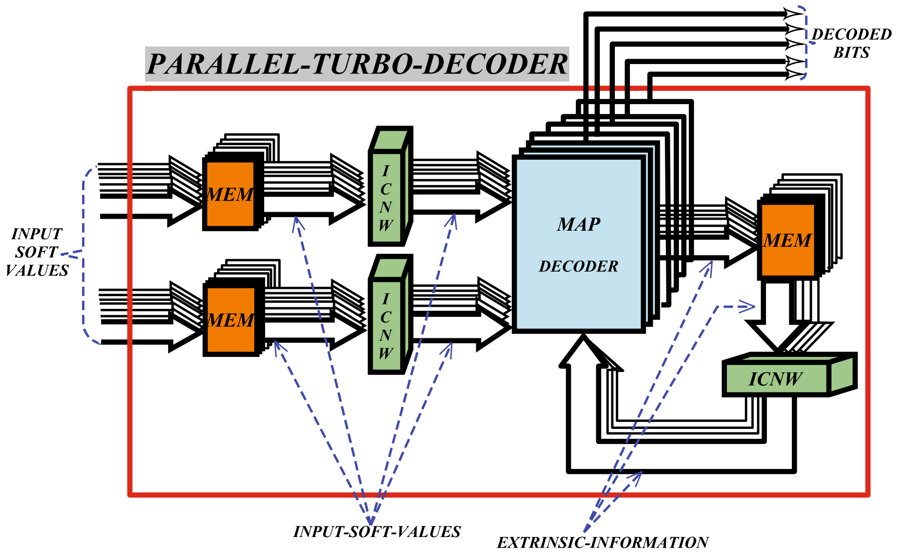

Fig. 1 A conventional parallel architecture of turbo decoder which iteratively processes input soft values to produce decoded bits

MAP algorithm are performed in [34] and [33], respectively. Our contributions in the design of memory-reduced architecture for parallel turbo decoder are as follows.

- New method of estimating the values of backward-recursion factors which initiate back trace in the sliding-window Bahl–Cocke–Jelinek–Raviv (SWBCJR) algorithm [5] for MAP decoder is presented, and it is referred as the reduced sliding window maximum a posteriori probability (RSWMAP) algorithm. Furthermore, a branch-metric-reformulation technique is provided to reduce the memory requirement.
- An architecture of MAP decoder based on RSWMAP algorithm and branch-metric reformulation is presented. Scheduling of this new MAP decoder and a comparative analysis of memory consumption by the proposed and conventional MAP-decoder-based parallel turbo decoders are carried out.
- Simulations for the BER performances of MAP and parallel turbo decoders are accomplished. An overall hardware saving of the turbo decoder with parallel architecture, based on RSWMAP algorithm and branch-metric reformulation, is estimated. Finally, the savings of hardware resources due to reduced-memory architecture of this work are compared with the reported contributions.

Outline The remainder of this paper is organized as follows. In Sect. 2, brief discussion on the mathematical background of Bahl–Cocke–Jelinek–Raviv (BCJR) algorithm is carried out. Sect. 3 presents detail explanation of the suggested RSWMAP algorithm and branch-metric-reformulation technique. In Sect. 4, architectural and scheduling details of the MAP decoder architecture are included. Section 5 presents BER performance evaluation of the MAP and turbo decoders, implementation details and comparison with the reported works. Finally, this paper is concluded in Sect. 6.

2 Theoretical Background

The BCJR algorithm determines probability, which is represented as P($U_t|y$), that the transmitted bit U_t is 1/0 provided the sequence y of corrupted soft values is received [4]. This is equivalent to the a posteriori logarithmic likelihood ratio $\Gamma(U_t|y)$, which is obtained by the logarithmic transformation of probability ratio as

$$\Gamma(U_t|y) = \ln\left\{\frac{P(U_t = 1|y)}{P(U_t = 0|y)}\right\}, \tag{1}$$

for hardware-efficient implementation of conventional BCJR algorithm [11]. The sign of $\Gamma(U_t|y)$ indicates whether the transmitted bit is 1/0, and its magnitude indicates the likelihood of determining a correct value of the transmitted bit. If $(s', s)\rightarrow 1$ and $(s', s) \rightarrow 0$ represent the sets of state transitions for the transmitted bit $U_t = 1$ and $U_t = 0$, respectively, then $\Gamma(U_t|y)$ can be expressed as [39]

$$\Gamma(U_t|y) = \ln\left\{\frac{\sum_{(s',s)\rightarrow 1} P(s', s, y)}{\sum_{(s',s)\rightarrow 0} P(s', s, y)}\right\}. \tag{2}$$

The corrupted-received sequence y can be partitioned into three subparts: $y_{i<t-1}$, $y_{i=t}$ and $y_{i>t+1}$. Such that $y_{i=t}$ represents the part of y received at an instant t and the other two parts of y received before and after this instant are $y_{i<t-1}$ and $y_{i>t+1}$, respectively. Thereby, the probability $P(s', s, y)$ from (2) can be expressed as

$$P\left(s', s, y\right) = P\left(s', s, y_{i<t-1}, y_{i=t}, y_{i>t+1}\right). \tag{3}$$

Applying Bayes' rule and assuming that the channel is memory less and discrete [39], an expression for $P(s', s, y)$ from (3) can be rewritten as

$$\begin{aligned} P\left(s', s, y\right) &= P\left(y_{i>t+1}|s\right) \times P\left(y_{i=t}, s|s'\right) \times P\left(y_{i<t-1}|s\right) \\ &= \beta_t\left(s\right) \times \gamma_t\left(s', s\right) \times \alpha_{t-1}\left(s'\right) \end{aligned} \tag{4}$$

where $\alpha_{t-1}(s')$, $\beta_t(s)$ and $\gamma_t(s', s)$ are forward-recursion factor, backward-recursion factor and branch metric, respectively. These entities are involved in the computation of $\Gamma(U_t|y)$ for successive trellis stages. Thereby, expression for $\Gamma(U_t|y)$ can be formulated by substituting the value of $P(s', s, y)$ from (4) in (2) as

$$\Gamma(U_t|y) = \ln\left\{\frac{\sum_{(s',s)\rightarrow 1} \alpha_{t-1}(s') \times \gamma_t\left(s', s\right) \times \beta_t(s)}{\sum_{(s',s)\rightarrow 0} \alpha_{t-1}(s') \times \gamma_t\left(s', s\right) \times \beta_t(s)}\right\} \quad \forall \quad t = \{1, 2, 3 \ldots N\} \tag{5}$$

where N represents the block length of code. This logarithmic value of $\Gamma(U_t|y)$ is computed for each trellis stage using forward-recursion and backward-recursion factors of all trellis states and branch metrics associated with all state transitions. Assuming that δ represents trellis transition where $s^{\mathrm{st}}(\delta)$ and $s^{\mathrm{en}}(\delta)$ correspond to start and end states, the logarithmic version of BCJR algorithm computes $\Gamma(U_t|y)$ value for tth trellis stage as

$$\Gamma(U_t|y) = \widehat{\max}_{\delta:(s',s)\Rightarrow 1}\{f(\delta)\} - \widehat{\max}_{\delta:(s',s)\Rightarrow 0}\{f(\delta)\} \tag{6}$$

based on the approximation using Jacobian logarithm [39] and the function $f(\delta)$ is expressed as

$$f(\delta) = A_{t-1}\left\{s^{\text{st}}(\delta)\right\} + Y_t(\delta) + B_t\left\{s^{\text{en}}(\delta)\right\} \tag{7}$$

where $A_{t-1}(s)$, $B_t(s)$ and $Y_t(s', s)$ are the logarithmic forms of forward-recursion factor, backward-recursion factor and branch metric, respectively, and their details are presented in Sect. 3.

3 Algorithmic Contributions

This section presents RSWMAP algorithm, with suggested method of estimating backward-recursion factors during the back trace, and mathematical reformulation of branch-metric equations.

3.1 RSWMAP Algorithm

In the conventional BCJR algorithm [4], computations of forward-recursion factors, backward-recursion factors and branch metrics for the entire trellis stages result in huge memory requirement and large decoding delay. Unlike this algorithm, the SWBCJR algorithm is a decoding strategy in which the entire trellis structure is segmented into number of sliding windows [5], and each window covers M trellis stages which is referred as sliding window size. This value of M affects memory requirement as well as decoding delay of the decoder, and its magnitude must be designed cautiously for the SWBCJR algorithm to achieve adequate error-rate performance. Simultaneously, initialization of backward-recursion factors while back tracing the trellis stages is an important factor responsible for the BER performance. The RSWMAP algorithm focusses on the estimation of backward-recursion factor values, which initiates the back trace, and aims to deliver better performance. On the other hand, forward-recursion factors and $\Gamma(U_t|y)$ values are computed by conventional methods in this algorithm. The major steps involved in the RSWMAP algorithm are presented as follows.

Initialization Assuming that the encoder is reset, the forward-recursion factors in the logarithmic version of SWBCJR algorithm are initialized as

$$\begin{aligned} A_{t=0}(s_k) &= ln\{\alpha_{t=0}(s_k)\} = ln(1) = 0 \quad \forall \quad k = 0. \\ A_{t=0}(s_k) &= ln\{\alpha_{t=0}(s_k)\} = ln(0) = -\infty \quad \forall \quad k \neq 0. \end{aligned} \tag{8}$$

Forward recursion The forward-recursion factor of each state for successive trellis stages is computed as

$$\alpha_t\left(s_k\right) = P\left(y_{i<t-1}|s_k\right) = \sum_{\psi:all\ s_k'} \alpha_{t-1}\left(s_k'\right) \times \gamma_t\left(s_k', s_k\right) \ \ \forall \ \ k\in S_N \tag{9}$$

based on (2) where the function ψ represents the transitions from all the previous states of $(t-1)$th trellis stage associated with the state s_k of tth trellis state, S_N is the total number of states in each trellis stage and $\gamma_t(s', s)$ is the branch metric which is expressed as

$$\gamma_t(s_k', s_k) = \exp\left\{U_t \times L(U_t)/2\right\} \times \exp\left(\frac{Lc}{2}\sum_{l=1}^{n} y_{tl} \times x_{tl}\right) \tag{10}$$

where Lc and n represent channel-reliability measure and code length, respectively. Similarly, $L(U_t)$ is the a priori probability of the transmitted information bit, x_{tl} and y_{tl} are transmitted bit and its received soft value, respectively. On the other hand, logarithmic versions of forward-recursion factor and branch metric are expressed as $A_t(s_k) = \ln\{\alpha_t(s_k)\}$ and $Y_t(s_k', s_k) = \ln\{\gamma_t(s_k', s_k)\}$, respectively, as discussed in Sect. 2. Thereby, this logarithmic version of forward-recursion factor is obtained by substituting the value of $\alpha_t(s_k)$ from (9) in the expression $A_t(s_k)$ and is given as

$$\begin{aligned} A_t\left(s_k\right) &= \ln\left[\sum_{\psi:all\ s_k'} \exp\left\{A_{t-1}\left(s_k'\right)\right\} \times \exp\left\{Y_t\left(s_k', s_k\right)\right\}\right] \\ &= \ln\left[\sum_{\psi:all\ s_k'} \exp\left\{A_{t-1}\left(s_k'\right) + Y_t\left(s_k', s_k\right)\right\}\right]. \end{aligned} \tag{11}$$

Thereby, based on Jacobian logarithmic approximation, which states that $\ln(\mathrm{e}^x + \mathrm{e}^y) \approx \max(x, y) + \ln(1 + \mathrm{e}^{-|x-y|}) = \widehat{\max}(x, y)$, the forward state metric can be approximated as

$$A_t(s_k) \approx \widehat{\max}_{\psi:alls_k'}\left\{A_{t-1}(s_k) + Y_t\left(s_k', s_k\right)\right\}. \tag{12}$$

Similarly, the branch metric computation by the logarithmic version of SWBCJR algorithm is carried out as

$$\begin{aligned} Y_t(s_k', s_k) &= \ln\{\gamma_t(s_k', s_k)\} = \ln\left[\exp(U_t \times L(U_t)/2) \times \exp\left(\frac{Lc}{2}\sum_{l=1}^{n} y_{tl} \times x_{tl}\right)\right] \\ Y_t(s_k', s_k) &= U_t \times L(U_t)/2 + \frac{Lc}{2}\sum_{l=1}^{n} y_{tl} \times x_{tl}. \end{aligned} \tag{13}$$

Back trace and estimation of backward recursion factor Conventional SWBCJR algorithm estimates backward-recursion factors based on border-metric initialization techniques [5] where if M represents sliding window size, then for $t \geq M$, that is

t={M, 2·M, 3·M, 4·M ......}, the border backward-recursion factors are initialized as $\beta_t(s_k) = 1/S_N \forall\, k = \{1, 2, 3\ \ldots\ S_N\}$ during the back trace. Thereby, the backward-recursion factors for successive trellis stages are computed from *t-1* to *t-M* as

$$\beta_t(s_k) = \sum_{\lambda:all\ \ s_k''} \beta_{t+1}(s_k) \times \gamma_{t+1}\left(s_k'', s\right)\ \ \forall\ \ k \in S_N \tag{14}$$

where the function λ represents the transitions from all the states of (t+1)th trellis stage associated with the state s_k of tth trellis state. Analogous to the logarithmic computation of forward-recursion metrics, the logarithmic version of backward-recursion factor is computed as

$$B_t(s_k) \approx \widehat{\max}_{\lambda:alls_k''}\left\{B_{t+1}\left(s_k''\right) + Y_t\left(s_k'', s_k\right)\right\}. \tag{15}$$

based on Jacobian logarithm, and the border backward-recursion factors are initialized as $B_t(s_k) = \ln(1/S_N)\ \forall\, k = \{1, 2, 3\ \ldots\ S_N\}\ \&\ t \geq M$.

This paper suggests a new method of initializing the border backward-recursion factors which initializes the back traces in SWBCJR algorithm. Consider a trellis graph that relates present, past and future trellis states at an instant t. This relation can be mathematically expressed by the a posteriori transition probability from (4) where the backward-recursion factor has been expressed as

$$\beta_t(s_k) = P(y_{i>t+1}|s_k). \tag{16}$$

It represents the probability that the received sequence after an instant t+1 is $y_{i>t+1}$ at s_k state. At t=M and using the above equation, the initial value of border backward-recursion factor responsible for initiating the back traces can be expressed as

$$\beta_M(s_k) = P\left(y_{i>M+1}|s_k\right) = \sum_{\lambda:all\ s_k''} P\left\{\left(y_{i>M+1}, s_k''\right)|s_k\right\} \tag{17}$$

where s_k'' represents a set of trellis states at $t = M$+1 and they are associated with the transitions to state s_k, during back trace. The probability from (17) can be further expressed as

$$\begin{aligned}\beta_M\left(s_k\right) &= \sum_{\lambda:all\ \ s_k''} P\{\left(y_{i=M}, y_{i>M}, s_k''\right)|s_k\} = \sum_{\lambda:all\ \ s_k''} P\left[\{(y_{i>M}), \left(y_{i=M}, s_k''\right)\}|s_k\right]\\ &= \sum_{\lambda:all\ \ s_k''} P\{(y_{i>M})\,|\left(y_{i=M}, s_k'', s_k\right)\} \times P\{\left(y_{i=M}, s_k''\right)|s_k\}\end{aligned} \tag{18}$$

based on the Bayes' rule which states that $P[(X, Y)|Z] = P[X|(Y, Z)] \times P(Y|Z)$. Applying conditions of discrete memoryless channel in (18), the mathematical expression for $\beta_M(s_k)$ is given as

$$\beta_M(s_k) = \sum_{\lambda:all\ \ s_k''} P(y_{i>M}|s_k) \times P\{(y_{i=M}, s_k'')|s_k\}. \tag{19}$$

Referring (4), the probabilities $P(y_{i>M}|s_k)$ & $P\{(y_{i=M}, s_k'')|s_k\}$ from (19) can be expressed as $\beta_{M+1}(s_k'')$ and $\gamma_M(s_k'', s_k)$, respectively. Finally, the value of estimated backward-recursion factor is

$$\beta_M(s_k) = \sum_{\lambda:all\ \ s_k''} \beta_{M+1}\left(s_k''\right) \times \gamma_M\left(s_k'', s_k\right) \tag{20}$$

where $\beta_{M+1}(s_k'')$ represents the probability of encoder state at an instant $t{=}M{+}1$ provided that the received sequence is $y_{i>M+1}$. This expression can be replaced by $1/S_N$ which is a probability that the encoder can attain one of the S_N states. Thereby, an expression for $\beta_M(s_k)$ from (20) can be restated as

$$\beta_M(s_k) = \frac{1}{S_N} \sum_{\lambda:all\ \ s''} \gamma_M\left(s_k'', s_k\right). \tag{21}$$

Subsequently, the logarithmic version of this suggested border backward-recursion factor can be expressed as

$$\begin{aligned} B_M(s_k) &= \ln\left(\frac{1}{S_N}\right) + \ln\left[\sum_{\lambda:all\ \ s''} \gamma_M\left(s_k'', s_k\right)\right] \\ &= \ln\left(\frac{1}{S_N}\right) + \widehat{\max}_{\lambda:alls''}\left\{\gamma_M\left(s_k'', s_k\right)\right\}. \end{aligned} \tag{22}$$

Unlike the conventional SWBCJR algorithm [5], which is based on border initialization method, dummy-backward-recursion-based SWBCJR algorithms perform an extra backward recursion for L_∂ trellis stages to estimate the value of backward-recursion factors at $t = M$ from the trellis stage $t = M + L_\partial$ [42]. Thereafter, the actual back trace commences from $t = M$ using the estimated backward-recursion factors from dummy back trace. Comparatively, this method has better coding performance in comparison with the conventional SWBCJR algorithm [5]. Thereby, the suggested method for initialization of backward-recursion factor from (22) can be extended for such SWBCJR algorithm based on dummy backward recursion as

$$B_{M+L_\partial}(s_k) = \ln\left(\frac{1}{S_N}\right) + \widehat{\max}_{\lambda:alls''}\left\{\gamma_{M+L_\partial}\left(s_k'', s_k\right)\right\}. \tag{23}$$

Computation of $\Gamma(U_t|y)$ Eventually, the logarithmic value of the a posteriori probability ratio $\Gamma(U_t|y)$ for tth trellis stage is computed using (6) and (7), as discussed earlier in Sect. 2, and is expressed as

$$\begin{aligned} \Gamma\left(U_t|y\right) = {} & \widehat{\max}_{\delta:(s_k', s_k)\Rightarrow 1}\left[A_{t-1}\left(s_k'\right) + Y_t\left(s_k', s_k\right) + B_t\left(s_k\right)\right] \\ & -\widehat{\max}_{\delta:(s_k', s_k)\Rightarrow 0}\left[A_{t-1}\left(s_k'\right) + Y_t\left(s_k', s_k\right) + B_t\left(s_k\right)\right]. \end{aligned} \tag{24}$$

Thereby, the decoded bit V_t is determined based on the magnitude of $\Gamma(U_t|y)$ such that

$$V_t = 1 \;\; \forall \;\; \Gamma(U_t|y) \geq 0; \qquad V_t = 0 \;\; \forall \;\; \Gamma(U_t|y) < 0. \tag{25}$$

3.2 Mathematical Reformulation of Branch-Metric Equations

Mathematical reformulation of branch-metric expression has been suggested in this paper to lower the memory requirement of MAP decoder. Considering a trellis graph for decoding the code generated by convolutional encoder with a transfer function of $\{1,(1+D+D^3)/(1+D^2+D^3)\}$ for $n = 2$, the branch-metric expression from (13) is expressed as

$$Y_t\left(s'_k, s_k\right) = \frac{1}{2} \times U_t \times L(U_t) + \frac{Lc}{2}\left(x_{t1} \times y_{t1} + x_{t2} \times y_{t2}\right) \tag{26}$$

where x_{t1} and x_{t2} are the systematic and parity bits, respectively, such that $x_{t1} \in \{+1, -1\}$ and $x_{t2} \in \{+1, -1\}$. Similarly, y_{t1} and y_{t2} are their respective soft values. It is to be noted that for a code length of n, each trellis stage needs at least 2^n branch metrics and they are referred as parent branch metrics [39]. Thereby, these parent branch metrics for $n = 2$ are expressed as below and their state transitions are shown in Fig. 2, which represents a trellis stage of trellis graph with eight states based on the considered encoder transfer function.

$$\begin{aligned} &Y_t(s'_0, s_0) = -\tfrac{1}{2} \times L\left(U_t\right) + \tfrac{Lc}{2}(-y_{t1} - y_{t2}),\\ &Y_t(s'_0, s_4) = \tfrac{1}{2} \times L\left(U_t\right) + \tfrac{Lc}{2}(y_{t1} - y_{t2}),\\ &Y_t(s'_4, s_2) = -\tfrac{1}{2} \times L\left(U_t\right) + \tfrac{Lc}{2}(-y_{t1} + y_{t2}) \text{ and}\\ &Y_t(s'_4, s_6) = \tfrac{1}{2} \times L\left(U_t\right) + \tfrac{Lc}{2}(y_{t1} + y_{t2}). \end{aligned} \tag{27}$$

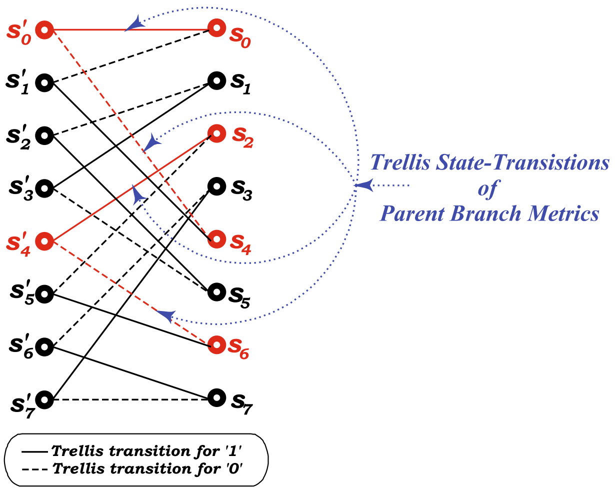
s'_0
s'_1
s'_2
s'_3
s'_4
s'_5
s'_6
s'_7
s_0
s_1
s_2
s_3
s_4
s_5
s_6
s_7
Trellis State-Transistions of Parent Branch Metrics
Trellis transition for '1'
Trellis transition for '0'

Fig. 2 A trellis stage of trellis graph with $S_N = 8$, $n{=}2$ and transfer function $\{1,(1+D+D^3)/(1+D^2+D^3)\}$

Among these parent branch metrics, $Y_t(s'_0, s_0)$ and $Y_t(s'_4, s_2)$ can be expressed using $Y_t(s'_4, s_6)$ and $Y_t(s'_0, s_4)$, respectively, as given below

$$\begin{aligned} Y_t\left(s'_0, s_0\right) &= -\left[\frac{1}{2} \times L\left(U_t\right) + \frac{Lc}{2}\left(y_{t1} + y_{t2}\right)\right] = -Y_t\left(s'_4, s_6\right). \\ Y_t\left(s'_4, s_2\right) &= -\left[\frac{1}{2} \times L\left(U_t\right) + \frac{Lc}{2}\left(y_{t1} - y_{t2}\right)\right] = -Y_t\left(s'_0, s_4\right). \end{aligned} \tag{28}$$

Reformulating the parent branch-metric expression of $Y_t(s'_0, s_0)$ from (27), the value $L(U_t) = -Lc(y_{t1} + y_{t2}) - 2 \times Y_t(s'_0, s_0)$ which is substituted in the second branch-metric expression of $Y_t(s'_4, s_2)$ from (28) and it simplifies to

$$\begin{aligned} Y_t(s'_4, s_2) &= Y_t\left(s'_0, s_0\right) + Lc \times y_{t2} = -Y_t\left(s'_0, s_4\right) \\ &\Rightarrow Y_t\left(s'_4, s_2\right) = -Y_t\left(s'_4, s_6\right) + Lc \times y_{t2} = -Y_t\left(s'_0, s_4\right), \end{aligned} \tag{29}$$

because $Y_t(s'_0, s_0) = -Y_t(s'_4, s_6)$ from (28). Referring the reformulated equations for parent branch metrics from (28) and (29), single parent branch metric $Y_t(s'_4, s_6)$ needs to be computed as well as stored for each trellis stage and the rest can be derived as

$$\begin{aligned} &Y_t\left(s'_0, s_0\right) = -Y_t\left(s'_4, s_6\right), \\ &Y_t\left(s'_0, s_4\right) = Y_t\left(s'_4, s_6\right) - Lc \times y_{t2},\ and \\ &Y_t\left(s'_4, s_2\right) = -Y_t\left(s'_4, s_6\right) + Lc \times y_{t2}. \end{aligned} \tag{30}$$

In design and implementation of MAP decoder based on conventional SWBCJR algorithm [42], it has to store 2^n parent branch metrics of each trellis stage for at least two sliding windows. Thereby, if n_γ represents the quantization bits of branch metric, then the MAP decoder architecture must accommodate a memory to store $2 \times M \times 2^n \times n_\gamma$ bits for parent branch metrics alone. Unlike the conventional method, the MAP decoder based on suggested branch-metric reformulation needs to store $2 \times M \times n_\gamma$ bits for the branch metrics. Assuming a case study where the values of $M = 32$, $n = 2$ and $n_\gamma = 8$ bits, the memory required for parent branch metrics using branch-metric reformulation is 75 % lesser than the memory required for parent branch metrics by conventional SWBCJR algorithm.

4 Architecture and Scheduling of Decoder

In this section, an architecture of MAP decoder and its scheduling based on RSWMAP algorithm and branch-metric reformulation are presented.

4.1 Decoder Architecture

Figure 3 shows suggested the MAP decoder architecture which is based on RSWMAP algorithm and branch-metric reformulation. This design is fed with input soft values (y_{t1} and y_{t2}) and the a priori probability $\{L(U_t)\}$. These values are processed by

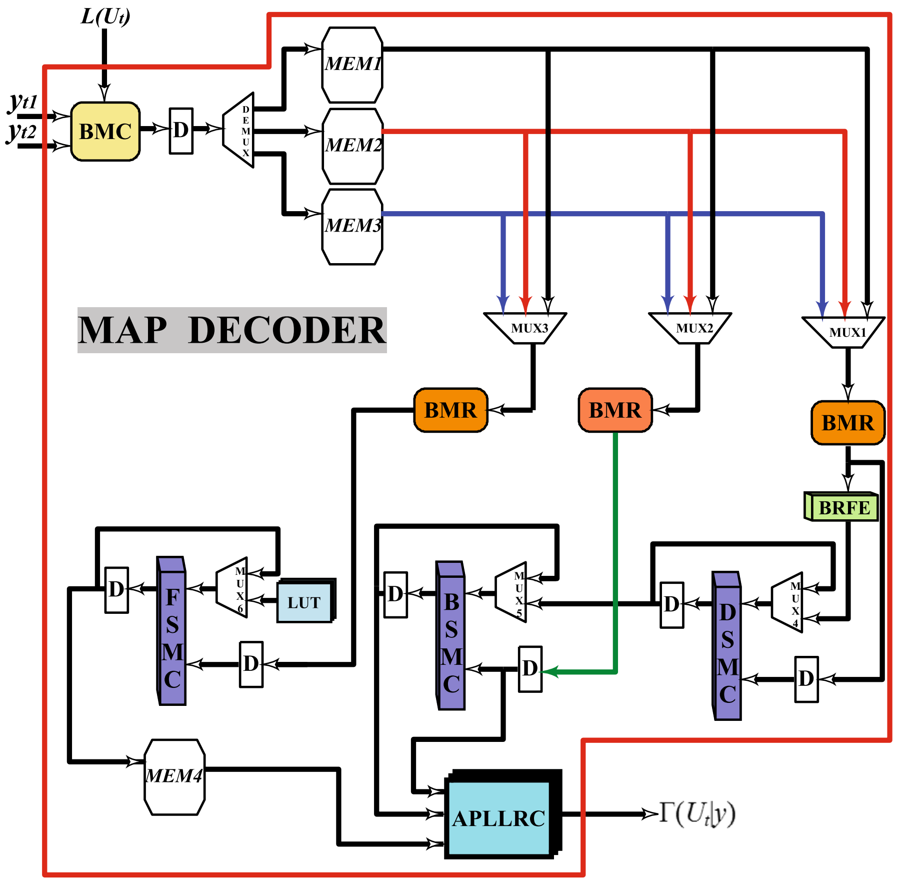

Fig. 3 High-level architecture of MAP decoder based on RSWMAP algorithm and branch-metric reformulation

branch-metric computation (BMC) submodule which calculates the value of $Y_t(s'_4, s_6)$, and it is used for computing rest of the parent branch metrics, as derived in (30). Corresponding architecture of BMC submodule is shown in Fig. 4a for computing the value of $Y_t(s'_4, s_6)$ assuming that the value of $Lc = 2$, which is sufficient for the MAP decoder to deliver adequate error-rate performance [11]. Its output is routed to three separate memories: *MEM1*, *MEM2* and *MEM3* via demultiplexer, as shown in Fig. 3, and each memory stores $M \times n_\gamma$ bits for M branch metrics $Y_t(s'_4, s_6)$. Outputs from these memories are multiplexed and are fed to the branch-metric router (BMR) submodule which has an architecture, as shown in Fig. 4b. It computes rest of the parent branch metrics $Y_t(s'_0, s_0)$, $Y_t(s'_0, s_4)$ and $Y_t(s'_4, s_2)$ from (30).

Figure 4(c) shows an architecture of backward-recursion factor estimator (BRFE) submodule which computes logarithmic version of estimated backward-recursion factors $B_M(s_k)\ \forall\ k = \{1, 2, 3, \ldots S_N\}$ that is expressed by (23). Branch metrics are the inputs for BRFE submodule and are fed to the comparators for determining the maximum values which are then added with a constant value of $\ln(1/S_N)$ from look-up-table

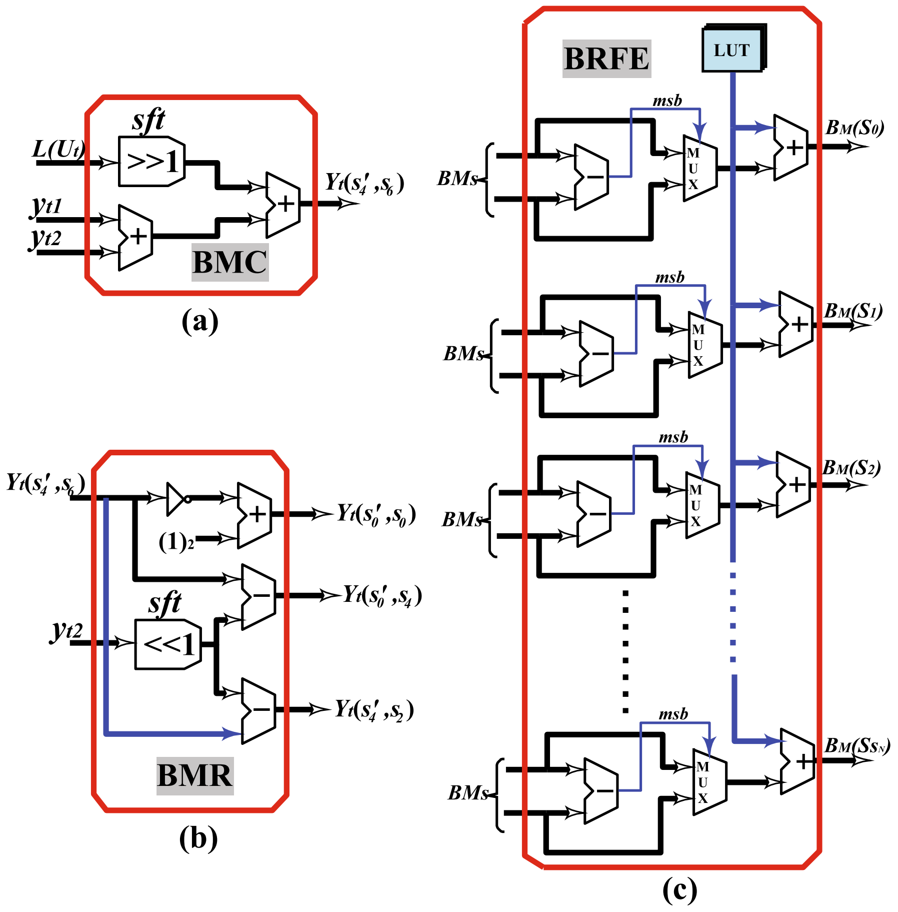

Fig. 4 Logic-level architectures of **a** branch-metric computation (BMC) submodule **b** branch metric router (BMR) submodule **c** backward-recursion factor estimator (BRFE) submodule

(LUT). In Fig. 3, the estimated backward-recursion factors from BRFE submodule are fed to dummy state metric computation (DSMC) submodule, which is used for dummy back trace while decoding with suggested RSWMAP algorithm. DSMC submodule is state metric computation (SMC) unit that comprises of S_N add-compare-select (ACS) units for computing backward-recursion factors for S_N states of a trellis stage [20]. It is fed with the branch metrics from BMR submodule and the multiplexed value of its output, which is fed back, along with the estimated backward-recursion factors from BRFE submodule, as shown in Fig. 3. Outputs from this DSMC submodule are fed to backward state metric computation (BSMC) submodule, which is also a SMC unit. It computes backward-recursion factors using branch metrics and dummy backward-recursion factors obtained from BMR and DSMC submodules, respectively, for successive trellis stages during the back trace. Another SMC unit with feedback architecture is forward state metric computation (FSMC) submodule for computing

S_N forward-recursion factors, while forward-recursion process in RSWMAP decoding, as shown in Fig. 3. During this process, the forward-recursion factors for the first trellis stage are initialized as $A_{t=0}(s_{k=0})$=0 and $A_{t=0}(s_{k\neq 0})$=$-\infty$ (practically very large negative number like -1). The computed values of forward-recursion factors by FSMC submodule are stored in the memory *MEM4*, which has a capacity to store $M\times S_N \times n_\alpha$ bits where n_α is the quantization of forward-recursion factor. Branch metrics obtained from BMR submodule, backward-recursion factors computed by BSMC submodule and fetched forward-recursion factors from *MEM4* are fed to the a posteriori logarithmic likelihood ratio computation (APLLRC) submodule. This combinational block determines the sum of $A_{t-1}(s'_k)$, $B_t(s_k)$ and $Y_t(s'_k, s_k)$ for all the state transitions and obtains respective maximum values among these sums for the transitions $(s'_k, s_k)\rightarrow 1$ and $(s'_k, s_k)\rightarrow 0$. Eventually, these maximum values are subtracted to give the value of $\Gamma(U_t|y)$ from (24), and then the decoded bits V_k are obtained consecutively from (25).

4.2 Decoder Scheduling

Scheduling for MAP decoding based on suggested RSWMAP algorithm is illustrated using timing chart from Fig. 5 which shows five essential processes involved in decod-

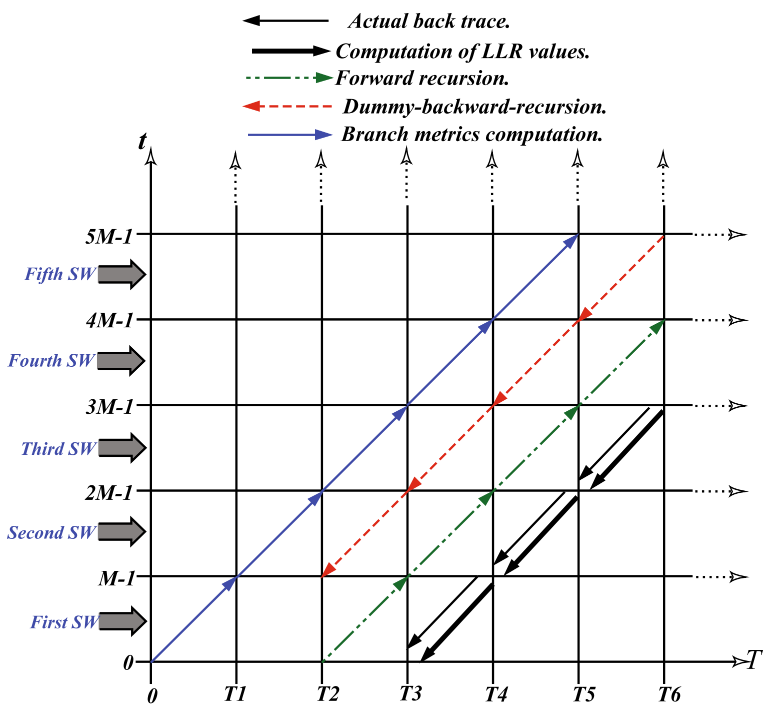

Fig. 5 Timing chart to illustrate scheduling of MAP decoding based on the suggested memory-reduced techniques

ing: forward-recursion, branch-metric computation, dummy back trace, back trace and computation of $\Gamma(U_t|y)$ performed along successive time slots of consecutive sliding windows while traversing through the trellis stages. Thereby, systematic explanation for the scheduling of RSWMAP-algorithm-based decoding is presented as follows using the timing chart from Fig. 5 for the suggested MAP decoder architecture.

- In the time slot $0 \leq T \leq T1$, the branch metrics $Y_t(s_4', s_6)\ \forall\ 0 \leq t \leq M-1$ are computed by BMC submodule for M trellis stages of first sliding window. These values are routed via demultiplexer to memory *MEM1* for storage.
- In the time slot $T1 \leq T \leq T2$, branch metrics $Y_t(s_4', s_6)\ \forall\ M-1 \leq t \leq 2M-1$ are computed for second sliding window and are stored in memory *MEM2*.
- In the time slot $T2 \leq T \leq T3$, computation of $Y_t(s_4', s_6)\ \forall 2M-1 \leq t \leq 3M-1$ for third sliding window along with its storage in *MEM3* and dummy back trace for the estimation of backward-recursion factors, which are used in the actual back trace of first sliding window, are carried out. This work assumes the value of $L_\delta = M$; thereby, initialization of backward-recursion factor $B_{M+L_\delta}(s_k) = B_{2M}(s_k)\ \forall\ k=\{1, 2, \ldots S_N\}$ is first accomplished using BRFE submodule of the decoder. Subsequently, dummy back trace is performed along the second sliding window till $t = M-1$ using DSMC submodule which is fed with the branch metrics fetched from *MEM2* and routed by BMR submodule. On the other hand, forward-recursion factors $A_t(s_k)\ \forall\ 0 \leq t \leq M-1$ and $k=\{1, 2, \ldots S_N\}$ are computed by FSMC submodule using branch metrics which are fetched from *MEM1* and then routed by BMR submodule.
- In the time slot $T3 \leq T \leq T4$, actual back trace for the computation of backward-recursion factors for the first sliding window ($0 \leq t \leq 2M-1$) is performed by BSMC submodule using the estimated values of backward-recursion factors from DSMC submodule and the branch metrics fetched from *MEM1*. These values of branch metrics from *MEM1*, computed backward-recursion factors of first sliding window and the forward-recursion factors those are fetched from *MEM4* and are fed as inputs to ALLRC submodule of the decoder. It uses these values to compute $\Gamma(U_t|y)$ from $t=M-1$ to $t=0$ for the first sliding window. It is to be noted that the memories *MEM1*, *MEM2*, *MEM3* and *MEM4* are dual-port random access memories (RAMs). Simultaneously, the dummy back trace for second sliding window is initiated from $t=3M$ and the branch metric of third sliding window is computed and stored in *MEM1*. Forward-recursion factors for the second sliding window is computed and then stored in *MEM4*.
- In the time slot $T4 \leq T \leq T5$, actual back trace for the computation of backward-recursion factors for the second sliding window ($M \leq t \leq 3M-1$) is performed by BSMC submodule using the estimated values of backward-recursion factors from DSMC submodule and the branch metrics fetched from *MEM2*. These values of branch metrics from *MEM2*, computed backward-recursion factors of second sliding window and the forward-recursion factors of second sliding window from *MEM4* are fed as inputs to ALLRC submodule. It uses these values to compute $\Gamma(U_t|y)$ from $t=2M-1$ to $t=\mathrm{M}$ for the second sliding window. Simultaneously, the dummy back trace for third sliding window is initiated from $t=4M$ and the branch metric of fourth sliding window is computed and stored in *MEM2*. Forward-

recursion factors for the third sliding window are computed and then stored in *MEM4*.

– This process of decoding successively continues until all the N values of $\Gamma(U_t|y)$ are obtained by MAP decoder.

4.3 Comparative Analysis of Memory Requirement

Scheduling of MAP decoder, as illustrated in timing chart of Fig. 5, has indicated that the decoder must store parent branch metric $Y_t(s'_4, s_6)$ for three SWs. This implies that the memories *MEM1*, *MEM2* and *MEM3* need to altogether store $3 \times M \times n_\gamma$ bits. Similarly, forward-recursion factors of M trellis stages where each stage has S_N states are stored in *MEM4* which has a size of $S_N \times M \times n_\alpha$ bits. Thereby, the total memory required by suggested MAP decoder architecture is

$$MEM_{\text{decoder}} = M \times (3\times n_\gamma + S_N \times n_\alpha) bits. \tag{31}$$

For a MAP decoder based on conventional SWBCJR algorithm with dummy backward recursion [42], the memory required for forward-recursion factors is same as that of the decoder presented in this work. However, such conventional MAP decoder has to store parent branch metrics for two sliding windows where each trellis stage has 2^n parent branch metrics; thereby, total of $M\times(2\times 2^n \times n_\gamma + S_N \times n_\alpha)$ bits are necessary to be stored. Similarly, MAP decoder which is designed using SWBCJR algorithm based on border initialization, as discussed in Sect. 3.1, needs to store parent branch metrics and forward-recursion factors for only one sliding window. Thus, the memory required by this decoder is $M\times(2^n \times n_\gamma + S_N \times n_\alpha)$ bits; however, it delivers degraded BER performance which defers such decoder from practical applications. On the other side, the conventional-BCJR-algorithm-based MAP decoder needs to store forward-recursion factors, backward-recursion factors and parent branch metrics for the entire N trellis stages [39]. Thereby, huge memory required by such MAP decoder is $N\times(S_N \times n_\alpha + S_N \times n_\beta + 2^n \times n_\gamma)$ bits where n_β is the quantization of backward-recursion factors. Thus, the hardware implementation of such decoder based on conventional BCJR algorithm is avoided in practice.

Parallel turbo decoder with multiple MAP decoders stores received probabilistic soft values of systematic and parity bits as well as the values for N extrinsic information, which are used in the iterative process of turbo decoding, as illustrated in Fig. 1. Thereby, Table 1 shows that the comparative analysis of memory required by such parallel turbo decoders based on four MAP algorithms suggested RSWMAP, dummy-backward-recursion-based SWBCJR, border-normalization-based SWBCJR and conventional BCJR algorithms. It shows that the memory required by soft values and extrinsic information of the turbo decoder is $N\times(n\times n_\varphi + n_\varepsilon)$ bits, which remains constant for all the parallel architectures of turbo decoder. In order to analyze the memory required by these parallel turbo decoders, plots of memory consumed (in bits) by turbo decoder with various parallel configurations of $P = 1, 4, 8, 16, 32$ and 64 number of MAP decoders are shown in Fig. 6. Subsequently, turbo decoder using border-normalization-based SWBCJR algorithm requires

Table 1 Comparison of the memory consumed by parallel turbo decoder based on different MAP algorithms

MAP algorithms	Required memory by turbo decoder (bit)
Proposed	$N\times(n\times n_\varphi + n_\varepsilon) + P\times M\times(3\times n_\gamma + S_N\times n_\alpha)$.
SWBCJR algorithm♠ [42]	$N\times(n\times n_\varphi + n_\varepsilon) + P\times M\times(2\times 2^n\times n_\gamma + S_N\times n_\alpha)$.
SWBCJR algorithm$^\flat$ [42]	$N\times(n\times n_\varphi + n_\varepsilon) + P\times M\times(2^n\times n_\gamma + S_N\times n_\alpha)$.
BCJR algorithm [39]	$N\times\{n\times n_\varphi + n_\varepsilon + P\times(S_N\times n_\alpha + S_N\times n_\beta + 2^n\times n_\gamma)\}$.

n_φ: quantization of input soft values of systematic and parity bits
n_ε: quantization of extrinsic information
P: total number of MAP decoders used in the parallel architecture of turbo decoder
♠: SWBCJR algorithm with dummy backward recursion
$\flat$: SWBCJR algorithm with border initialization method

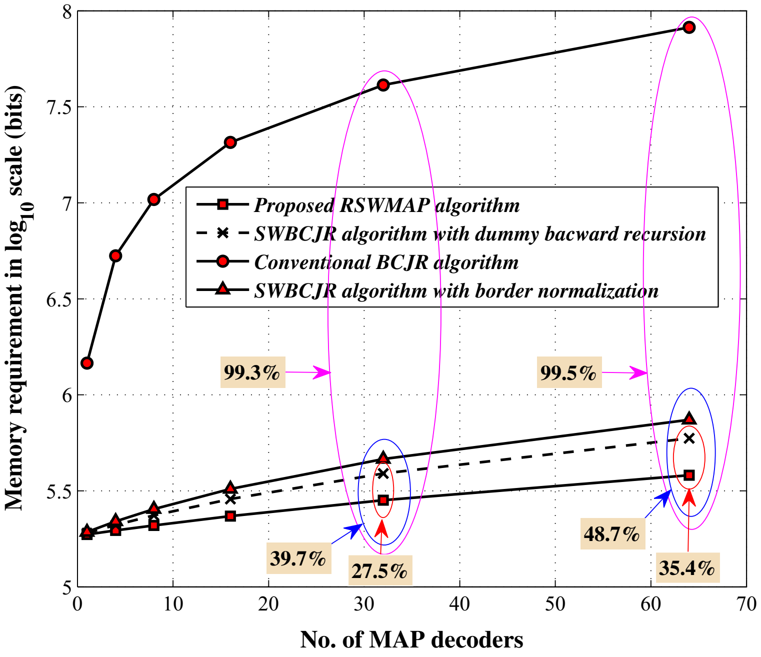

Fig. 6 Memory required by parallel turbo decoder architectures using branch-metric reformulation, SWBCJR- and BCJR algorithms-based MAP decoders. The plot is shown for the values $N = 6144$, $n = 3$, $S_N = 8$ and the quantization of $(n_\varepsilon, n_\varphi, n_\gamma, n_\alpha, n_\beta) = (9, 7, 8, 9, 9, 8)$ bits

nearly double the value of M, in comparison with other decoding algorithms considered in this paper, to deliver adequate BER performance. In Fig. 6, the values of M for SWBCJR algorithm based on border normalization and other decoding algorithms are 64 and 32, respectively. It can be observed that the proposed MAP-decoder-based design of turbo decoder requires the least number of bits to be stored, as compared to dummy-backward-recursion-based SWBCJR, border-normalization-based SWBCJR and conventional-BCJR-algorithms-based decoders. Subsequently, the percentages of improvements achieved by the suggested parallel turbo decoders with $P = 64$ and $P = 32$ configurations have been annotated in Fig. 6. For a turbo decoder with parallel architecture of $P = 64$, the proposed decoder with RSWMAP algorithm has shown 48.7 and 35.5 % of improvements in comparison with the decoders based on border-normalization-based SWBCJR and dummy-backward-recursion-based SWBCJR algorithms, respectively. Additionally, the suggested decoder consumes 99.5 % lesser memory as compared to decoder based on conventional BCJR algorithm, as shown in Fig. 6.

5 Performance Analysis, Trade-Offs and Comparison

In this section, BER performance analysis of MAP and parallel turbo decoders based on the suggested RSWMAP algorithm is carried out. From an implementation perspective, estimation of overall hardware saving achieved by parallel turbo decoders based on RSWMAP algorithm and branch-metric reformulation is presented. Finally, the overall memory saving of suggested decoder architecture is compared with the reported works.

5.1 BER Performance

Figure 7 shows the BER performance of MAP decoders, with a transfer function $\{1, (1 + D + D^3)/(1 + D^2 + D^3)\}$ and a code rate of 1/2, for additive white Gaussian noise (AWGN) channel using binary phase shift keying (BPSK) modulation scheme. This performance analysis is carried out for the MAP decoders based on RSWMAP, SWBCJR and BCJR algorithms with $M = 32$, using max-log-MAP approximation [39]. Figure 7 shows that a MAP decoder with RSWMAP algorithm performs better than the conventional SWBCJR-algorithm-based decoder by 1.28 dB at a BER of 10^{-5}. However, it has degraded performance of 0.21 dB, compared to BCJR-algorithm-based MAP decoder, at a BER of 10^{-5}. Similarly, the BER performance of parallel turbo decoder, in AWGN channel environment with BPSK modulation, for six decoding iterations and a turbo block length of 6144 bits is shown in Fig. 8. It shows that the BER performance of parallel turbo decoder based on RSWMAP algorithm for $M = 24$ has a coding gain of 0.4 dB at a BER of 10^{-4} in comparison with the decoder based on SWBCJR algorithm for the same value of $M = 24$. Subsequently, Fig. 8 shows that the SWBCJR-algorithm-based turbo decoder with $M = 32$ has a similar BER performance as the RSWMAP-algorithm-based turbo decoder decoder with $M = 24$.

5.2 Implementation Trade-Offs

Comparative study of BER performances has shown that the parallel turbo decoder based on RSWMAP algorithm achieves an adequate BER performance with smaller value of M, in comparison with the SWBCJR-algorithm-based parallel turbo decoder. A reduced sliding window size would require lesser memory for storing branch metrics and forward-recursion factors. The branch-metric reformulation as well as the RSWMAP algorithm contributes to memory saving in a MAP decoder. From the

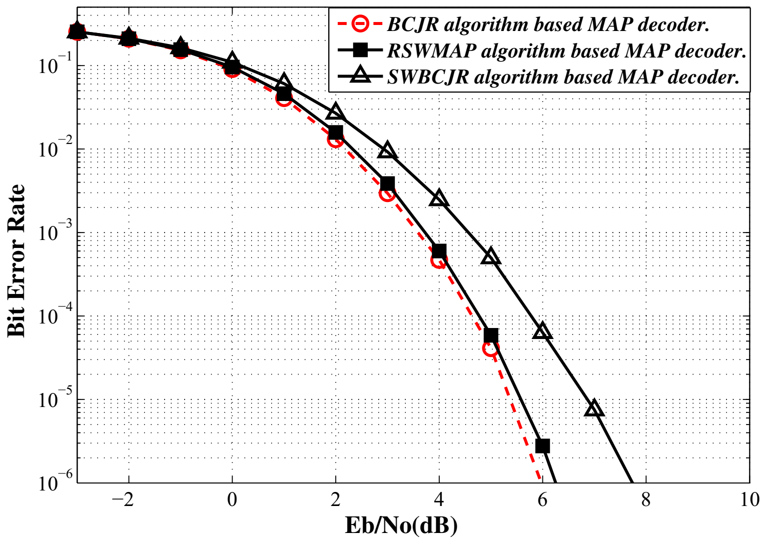

Fig. 7 BER performance of MAP decoders based on three MAP algorithms for a code rate of 1/2 and sliding window size of 32

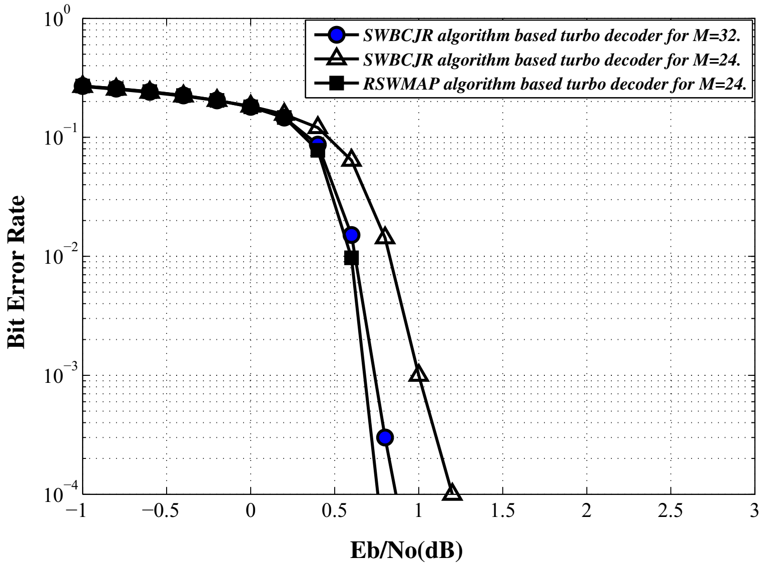

Fig. 8 BER performance of parallel turbo decoders with $P = 64$, based on different MAP algorithms for a code rate of 1/3 and six decoding iterations

implementation perspective, overall savings of hardware resources due to reduced-memory architecture of parallel turbo decoder, which uses MAP decoders based on branch-metric reformulation and RSWMAP algorithm, are presented here. Recently, the VLSI implementations of parallel turbo decoders with $P = 8$ [37], $P = 16$ [38],

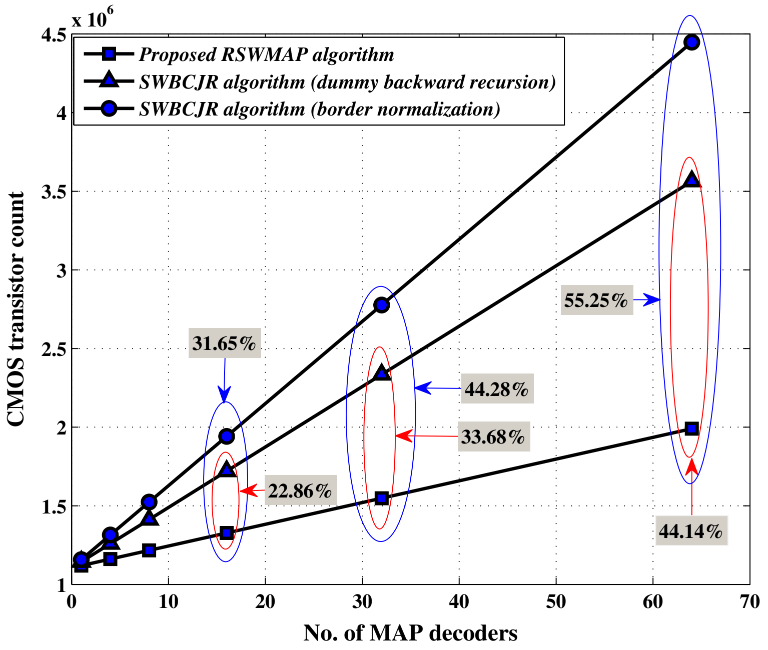

Fig. 9 Hardware savings in terms of CMOS transistor counts for parallel turbo decoders based on the proposed and the SWBCJR-algorithm-based MAP decoders

$P = 32$ [12] and $P = 64$ [29] have been reported for high-data-rate application. Thereby, the hardware savings of parallel turbo decoders are analyzed up to $P = 64$ parallel configuration. Such savings of parallel turbo decoders based on proposed algorithm are accounted in terms of complementary metal oxide semiconductor (CMOS) transistor count, and the comparison is carried out with parallel turbo decoders based on SWBCJR algorithms which is based on dummy backward recursion and border normalization. Assuming that the memory used in parallel turbo decoder is static random access memory (SRAM), it requires six CMOS transistors to store each bit [40].

Referring the expressions from Table 1, the parallel turbo decoders based on proposed algorithm and conventional SWBCJR algorithm which is based on dummy backward recursion consume $6\times\{N\times(n\times n_\varphi + n_\varepsilon) + P\times M\times(3\times n_\gamma + S_N\times n_\alpha)\}$ transistors and $6\times\{N\times(n\times n_\varphi + n_\varepsilon) + P\times M\times(2^{n+1}\times n_\gamma + S_N\times n_\alpha)\}$ transistors, respectively. Similarly, conventional SWBCJR algorithm based on border normalization consumes $6\times\{N\times(n\times n_\varphi + n_\varepsilon) + P\times M\times(2^n\times n_\gamma + S_N\times n_\alpha)\}$ transistors. Figure 9 shows the overall hardware savings in terms of CMOS transistor count for various parallel configuration of the decoder. From the previous BER analysis, it has been seen that parallel turbo decoder based on RSWMAP algorithm can deliver optimum BER performance for $M = 24$ rather than for $M = 32$, which is required by SWBCJR algorithm based on dummy backward recursion. On the other hand, parallel turbo decoder using SWBCJR algorithm based on border normalization would require at least $M = 64$ to deliver adequate BER performance. Thereby, Fig. 9 shows the CMOS transistors consumed by turbo decoders based on suggested MAP decoder for $M = 24$, SWBCJR-algorithm-based MAP decoder using dummy backward recursion for $M = 32$ and SWBCJR-algorithm-based MAP decoder using border normalization for $M = 64$. The percentage of hardware saving for different values of P, such as 16, 32 and 64, are shown in Fig. 9. Thereby, maximum of 52.25 and 44.14 % hardware resources are saved due to the reduction in memory in parallel turbo decoder for $P = 64$. However, decoder which uses SWBCJR algorithm based on border normalization is rarely used due to its degraded BER performance and poor hardware utilization.

5.3 Comparisons

As discussed in Sect. 3.2, memory saving of 75 % has been achieved by MAP decoder in this work, for the storage of branch metrics, in comparison with conventional SWBCJR-algorithm-based MAP decoder [42]. The overall saving of hardware resources in MAP decoder, due to reduced-memory architecture for forward/backward recursion factors and branch metrics altogether, is referred as state branch memory saving (SBMS). Figure 10 shows the percentages of SBMSs achieved by proposed and reported MAP architectures. Architecture *Arch-1* presented in [33] has achieved a saving due to reduced memory required for forward-recursion factors, and its SBMS is 50 %. Similarly, the MAP decoder kernel *Arch-2* designed in [34] has achieved a

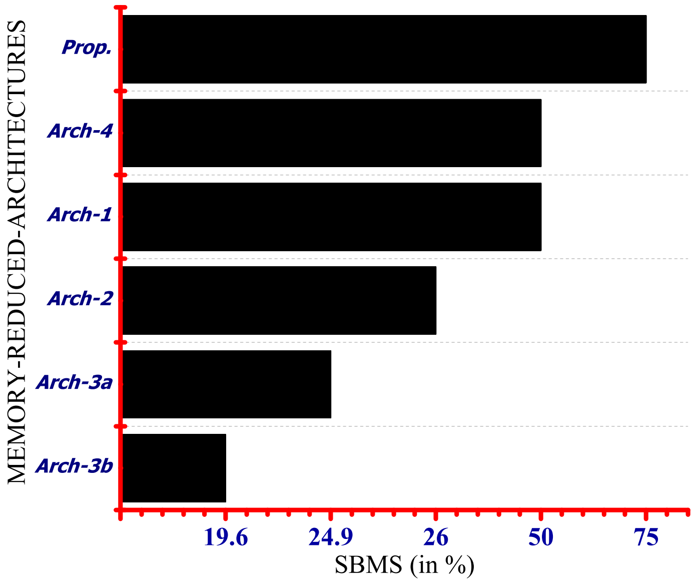

Fig. 10 Comparison for the state branch memory savings (SBMSs) of proposed and reported MAP decoder architectures w.r.t conventional MAP decoder: *Arch-1* [33], *Arch-2* [34], *Arch-3a* [17], *Arch-3b* [17] and *Arch-4* [24]

SBMS of 26 %. Low-power and reduced-memory architecture proposed in [17] has shown SBMSs of 24.9 and 19.6 % for radix-2 (*Arch3a*) and radix-4 (*Arch3b*) architectures, respectively. The state-metric-compression-based architecture *Arch4* of MAP decoder [24] has a SBMS of 50 %, as shown in Fig. 10. Thus, the memory-reduced architecture presented in this work has shown better SBMS in comparison with the reported architectures.

6 Conclusion

This paper has highlighted a new method of estimating backward-recursion factors to initiate the back trace for successive sliding windows in MAP algorithm. Another contribution was the mathematical reformulation of branch-metric equations, and this aided MAP decoder to store only single branch metric in each trellis stage. Based on these methods, architecture and scheduling of the MAP decoder were presented. Thereafter, comparative study on BER performance of parallel turbo decoders based on the proposed and conventional methods was carried out, and the former had a coding gain of 0.4 dB at a BER of 10^{-4}. The parallel turbo decoder with proposed MAP decoders has resulted in better coding performance and reduced-memory design. Finally, an overall hardware saving of this decoder was analyzed in terms of CMOS transistor count and it has shown that a saving of 44.14 % is possible for the case of 64 parallel MAP decoders. For the next generation wireless communication, higher data rates are needed and such hardware-efficient parallel turbo decoders are required to maintain the application of turbo code for evolving wireless communication standards.

References

1. C. Benkeser, Power efficiency and the mapping of communication algorithms into VLSI. Masters thesis, ETH Zurich, Switzerland, (2010)
2. C. Berrou, A. Glavieux, P. Thitimajshima, Near Shannon limit error-correcting coding and decoding: turbo-codes, in *Proceedings of the International Conference Communications*, pp. 1064–1070 (1993)
3. C. Benkeser, A. Burg, T. Cupaiuolo, Q. Huang, Design and optimization of an HSDPA turbo decoder ASIC. IEEE J. Solid State Circuits **44**, 98–106 (2009)
4. L. Bahl, J. Cocke, F. Jelinek, J. Raviv, Optimal decoding of linear codes for minimizing symbol error rate. IEEE Trans. Inf. Theory **20**, 284–287 (1974)
5. S. Benedetto, D. Divsalar, G. Montorsi, F. Pollara, Soft-output decoding algorithms in iterative decoding of turbo codes. in *Report JPL TDA Progress*, pp. 42–124 (1996)
6. M. Cheol, I.-C. Park, SIMD processor-based turbo decoder supporting multiple third-generation wireless standards. Journal **15**, 801–810 (2007)
7. R. Dobkin, M. Peleg, R. Ginosar, Parallel VLSI architecture for MAP turbo decoder, in Proceedings of the *IEEE International Symposium Personal, Indoor Mobile Radio Communication*, pp. 15–18 (2002)
8. G.D. Forney Jr, The viterbi algorithm. Proc. IEEE **61**, 268–278 (2005)
9. J.F. Hayes, T.M. Cover, J.B. Riera, Optimal sequence detection and optimal symbol-by-symbol detection: similar algorithms. IEEE Trans. Commun. **30**, 152–157 (1982)
10. J. Hagenauer, P. Hoeher, A Viterbi Algorithm with Soft-Decision Outputs and its Applications, Proceedings of the *3rd IEEE Global Telecommunication Conference*, Dallas, TX, pp. 1680–1686 (1989)
11. L. Hanzo, T.H. Liew, B.L. Yeap, *Turbo Coding, Turbo Equalisation and Space-Time Coding for Transmission over Fading Channels* (Wiley, England, 2003)
12. S.M. Karim, I. Chakrabarti, High throughput turbo decoder using pipelined parallel architecture and collision free interleaver. IET Commun. **6**, 1416–1424 (2012)

13. S. Kumawat, R. Shrestha, N. Daga, R.P. Paily, High-throughput LDPC-decoder architecture using efficient comparison techniques and dynamic multi-frame processing schedule. IEEE Trans. Circuits Syst. I Regul. Pap. **62**, 1421–1430 (2015)
14. J.P. Kulkarni, K. Kim, K. Roy, A 160 mV robust schmitt trigger based subthreshold SRAM. IEEE J. Solid State Circuits **42**, 2303–2313 (2007)
15. L. Lee, Real-time minimal-bit-error probability decoding of convolutional codes. IEEE Trans. Commun. **22**, 146–151 (1974)
16. C.-C. Lin, Y.-H. Shih, H.-C. Chang, C.-Y. Lee, A low power turbo/viterbi decoder for 3GPP2 applications. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **14**, 426–430 (2006)
17. C.-H. Lin, C.-Y. Chen, T.-H. Tsai, A.-Y. Wu, Low-power memory-reduced traceback MAP decoding for double-binary convolutional turbo decoder. IEEE Trans. Circuits Syst. I Reg. Pap. **56**, 1005–1016 (2009)
18. Y. Li, B. Vucetic, Y. Sato, Optimum soft-output detection for channels with intersymbol interference. IEEE Trans. Inf. Theory **41**, 704–713 (1995)
19. LTE: Evolved Universal Terrestrial Radio Access (E-UTRA): *Multiplexing and Channel Coding* (3GPP TS 36.212 version 10.0.0 Release 10) (2008)
20. G. Masera, G. Piccinini, M.R. Roch, M. Zamboni, VLSI architectures for turbo codes. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **7**, 369–379 (1999)
21. G. Masera, M. Mazza, G. Piccinini, F. Viglione, M. Zamboni, Architectural strategies for low-power VLSI turbo decoders. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **10**, 279–285 (2002)
22. H. Michel, A. Worm, N. Wehn, Influence of quantization on the bit-error performance of turbo-decoders. Proc. IEEE Veh. Technol. Conf. **1**, 581–585 (2000)
23. X. Ma, A. Kavcic, Path partitions and forward-only trellis algorithms. IEEE Trans. Inf. Theory **49**, 38–52 (2003)
24. M. Martina, G. Masera, State metric compression techniques for turbo decoder architectures. IEEE Trans. Circuits Syst. I Regul. Pap. **58**, 1119–1128 (2011)
25. J.G. Proakis, *Digital Communications, Reading*, 2nd edn. (McGraw-Hill, New York, 1989)
26. C.E. Shannon, A mathematical theory of communication: part-I. Bell Syst. Tech. J. **21**, 379–423 (1948)
27. C.E. Shannon, A mathematical theory of communication: part-II. Bell Syst. Tech. J. **21**, 623–656 (1948)
28. C. Studer, C. Benkeser, S. Belfanti, Q. Huang, Design and implementation of a parallel turbo-decoder ASIC for 3GPP-LTE. IEEE J. Solid State Circuits **46**, 8–17 (2011)
29. Y. Sun, J.R. Cavallaro, efficient hardware implementation of a highly-parallel 3gpp lte/lte-advance turbo decoder. Integr. VLSI J. **44**, 305–315 (2011)
30. Y. Sun, Y. Zhu, M. Goel, J.R. Cavallaro, Configurable and scalable high throughput turbo decoder architecture for multiple 4G wireless standards, in *International Conference on Applied-Specific System, Architecture and Processors*, pp. 209–214 (2008)
31. R. Shrestha, R.P. Paily, High-throughput turbo decoder with parallel architecture for LTE wireless communication standards. IEEE Trans. Circuits Syst. I Regul. Pap. **61**, 2699–2710 (2014)
32. S. Talakoub, L. Sabeti, B. Shahrrava, M. Ahmadi, An improved Max-Log-MAP algorithm for turbo decoding and turbo equalization. IEEE Trans. Instrum. Meas. **56**, 1058–1063 (2007)
33. T.-H. Tsai, C.-H. Lin, A new memory-reduced architecture design for Log-MAP algorithm in turbo decoding, in *IEEE 6th CAS Symposium on Emerging Technologies: Mobile and Wireless Communications*, vol 2, pp. 607–610 (2004)
34. T-H. Tsai, C-H. Lin, A.-Y. Wu, A memory-reduced Log-MAP kernel for turbo decoder, in *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol 2, pp. 1032–1035 (2005)
35. F. Vasefi, Z. Abid, Low power n-bit adder and multiplier using lowest-number-or-transistor 1-bit adders, in *Canadian Conference on Electrical and Computer Engineering*, pp. 1731–1734 (2005)
36. C.-C. Wong, H.-C. Chang, High-efficiency processing schedule for parallel turbo decoders using QPP interleaver. IEEE Trans. Circuits Syst. I Regul. Pap. **58**, 1412–1420 (2011)
37. C.-C. Wong, H.-C. Chang, Reconfigurable turbo decoder with parallel architecture for 3GPP LTE system. IEEE Trans. Circuits Syst. II Exp. Briefs **57**, 566–570 (2010)
38. C.-C. Wong, M.-W. Lai, C.-C. Lin, H.-C. Chang, C.-Y. Lee, Turbo decoder using contention-free interleaver and parallel architecture. IEEE J. Solid State Circuits **45**, 422–432 (2010)
39. J.P. Woodard, L. Hanzo, Comparative study of turbo decoding techniques: an overview. IEEE Trans. Veh. Technol. **49**, 2208–2233 (2000)

40. N.H.E. Weste, D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective, Reading* (Pearson-Addison Wesley, MA, 2005)
41. Y. Wu, B.D. Woerner, T.K. Blankenship, Data width requirements in SISO decoding with modulo normalization. IEEE Trans. Commun. **49**, 1861–1868 (2001)
42. Z. Wang, Z. Chi, K.K. Parhi, Area-efficient high-speed decoding scheme for turbo decoders. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **10**, 902–912 (2002)