A SORTING-BASED APPROXIMATION OF THE SUM–PRODUCT ALGORITHM

Xiao-Yu Hu and Thomas Mittelholzer

Abstract - A complexity- and delay-efficient simplification of the sum–product algorithm (SPA) for decoding low-density parity-check (LDPC) codes is presented. The key feature of the new algorithm consists of a modification of the complexity-intensive and delay-causing update equations at the check nodes of the factor graph of the LDPC code. The modified update equations at a check node are based on ordering the reliability values of the incoming messages and on using a balanced tree topology to achieve optimum parallel processing. Furthermore, the complexity of the new algorithm can be adjusted: the least complex version of the algorithm corresponds to the so-called min-sum approximation, and the most complex version gives the full SPA.

Keywords: sum–product algorithm, min-sum algorithm, low-density parity-check codes

Resumo - Uma simplificação eficiente, em termos de complexidade e de atraso, do algoritmo soma-produto (SPA) para a decodificação de códigos de verificação de paridade de baixa densidade (LDPC) é apresentada. A característica principal do novo algoritmo consiste na modificação das equações de atualização nos nós de verificação do gráfico fator do código LDPC, que são bastante complexas e causam atraso. As equações de atualização modificadas em um nó de verificação são baseadas no ordenamento dos valores de confiabilidade das mensagens que chegam e na utilização de uma topologia de árvore balanceada para se alcançar o processamento paralelo ótimo. Além disso, a complexidade do novo algoritmo pode ser ajustada: a versão menos complexa do algoritmo corresponde à assim chamada aproximação "min-sum", e a versão mais complexa ao SPA completo.

Palavras-chave: Algoritmo soma-produto, algoritmo "min-sum", códigos de verificação de paridade de baixa densidade

1. INTRODUCTION

Owing to their outstanding performance, low-density parity-check (LDPC) codes [1], are currently recognized to be the best class of codes to approach the Shannon limit efficiently, using iterative decoding [2]. Efficient implementation of the decoding algorithm in hardware has become an area of increased interest (see e.g. [3, 4] and references therein). In particular, substantial reductions of the total decoding complexity have been obtained by considering simplifications of the core operations in the sum–product algorithm (SPA).

Here we focus on various binary tree representations associated with the check nodes in the factor graph of an LDPC code. We propose a modification of the highly complex and delay-causing update equations at the check nodes that is based on ordering the reliability values of the incoming messages. Specifically, these messages are split into the set of a few least reliable messages and the set of all other messages, which are treated as being fully reliable. In this way, the complexity of the update equations can essentially be reduced to the case of only a few incoming messages. This reduction in complexity is most pronounced for high-rate LDPC codes, where each check node is connected to a large number of symbol nodes. The proposed simplification does not rely on reduced-complexity approximations of the core operations; therefore, one can achieve additional complexity reductions by applying such approximate core operations [3, 4, 5].

A critical performance issue of all turbo-like codes is the decoding delay that is inherent in iterative decoding of block codes. Following [4, 6], we devote special attention to low-delay implementations, which are based on balanced tree topologies for maximum parallel processing.

The paper is organized as follows. In Section 2, the SPA is reviewed, and the complexity of the different parts discussed briefly. In Section 3, different implementations for the check-node updates are considered in terms of special factor graphs that are binary trees. Section 4 is devoted to the use of ordered statistics of the incoming messages at the check nodes and to simplifications of the factor graph, both of which result in a reduction of the delay as well as of the total complexity. In Section 5, simulation results comparing the full SPA with the proposed simplified versions are shown. Finally, in Section 6, the key features of the proposed simplified algorithm are summarized.

2. THE SUM–PRODUCT ALGORITHM IN THE LOG-LIKELIHOOD DOMAIN

The SPA is an efficient iterative algorithm for decoding LDPC codes. In particular, given a received word $\mathbf{y} = [y_1, y_2, \ldots, y_N]$, which corresponds to some transmitted binary codeword $[x_1, x_2, \ldots, x_N]$, the SPA updates in each iteration step its "temporal beliefs" about the components x_n of the transmitted codeword given $\mathbf{y}$, and, eventually, makes a decision as to which codeword components were sent.

We describe the SPA in the log-likelihood domain using similar notations as in [7]. For a given binary sparse parity-check matrix H, we denote by $\mathcal{M}(n)$ the set of check nodes that are connected to symbol node n, i.e., the set of indices that are "1" in the n-th column of the parity-check

Xiao-Yu Hu and Thomas Mittelholzer are with IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland (E-mails: {xhu,tmi}@zurich.ibm.com.)

The results of this paper have been presented at ITS'02 in Natal, Brazil.

matrix H. Similarly, we denote by $\mathcal{L}(m)$ the set of symbol nodes that participate in the m-th parity-check equation, i.e., the set of indices that are "1" in the m-th row of H. We denote by $\mathcal{L}(m)\backslash n$ the set-theoretic difference of $\mathcal{L}(m)$, from which the n-th symbol node is excluded. Similarly, for the check-node sets, we introduce the set-theoretic difference $\mathcal{M}(n)\backslash m$. We denote by $L(q_{n\to m})$ the message that symbol node n sends to check node m, which corresponds to the log-likelihood ratio (LLR) of some temporal likelihood that the n-th symbol is "0" or "1". Similarly, $L(r_{m\to n})$ denotes the message that the m-th check node sends to the n-th symbol, which again is an LLR expressing the temporal belief that the m-th check node has about the n-th symbol being "0" or "1". The SPA that is based on log-likelihood messages can be stated as follows.

1. **Initialization**

   Each symbol node n is assigned an initial LLR $L(p_n)$. In the case of equiprobable inputs on a memoryless AWGN channel,

   $$L(p_n) = \frac{2}{\sigma^2} y_n.$$

   where the σ^2 is the noise variance, and y_n the corrupted value of input symbol x_n. For every incidence pair (m, n), i.e., pair of indices satisfying $H_{m,n} = 1$, the messages $L(q_{n\to m})$ and $L(r_{m\to n})$ are initialized as follows:

   $$\begin{aligned} L(q_{n\to m}) &= L(p_n), \\ L(r_{m\to n}) &= 0. \end{aligned}$$

2. **Check-node update**

   Each check node m gathers the incoming messages $L(q_{n\to m})$, and updates the belief on symbol n based on the information from all other bits connected to check node m:

   $$L(r_{m\to n}) = 2\tanh^{-1}\left\{ \prod_{n'\in\mathcal{L}(m)\backslash n} \tanh[L(q_{n'\to m})/2] \right\}. \quad (1)$$

   The outgoing message $L(r_{m\to n})$ is referred to as extrinsic information because it does not depend on the incoming message $L(q_{n\to m})$.

3. **Symbol-node update**

   Each symbol node n propagates its likelihood information to all the check nodes that connect to it. The outgoing messages are calculated as

   $$L(q_{n\to m}) = L(p_n) + \sum_{m'\in\mathcal{M}(n)\backslash m} L(r_{m'\to n}).$$

4. **Decision**

   The decoder obtains the total temporal *a-posteriori* information for symbol n by summing the likelihoods from all the check nodes that connect to bit n:

   $$\lambda_n = L(p_n) + \sum_{m\in\mathcal{M}(n)} L(r_{m\to n}).$$

The algorithm iterates until a valid codeword has been found, i.e., the hard decision of the temporal *a-posteriori* vector $[\lambda_1, \ldots, \lambda_N]$ satisfies the parity-check matrix H, or a preset maximum number of iterations has been reached.

Obtaining an efficient implementation of the symbol-node updates is obvious. By forming first the *a-posteriori* information λ_n, the extrinsic information terms are given by $L(q_{n\to m}) = \lambda_n - L(r_{m\to n})$. Thus, the total computational load for a symbol-node update is only $2|\mathcal{M}(n)|$ additions, where the cardinality $|\mathcal{M}(n)|$ denotes the degree of the variable node n.

The check-node update is the most complex part of the SPA. Two issues influence its computational complexity: the topology of the multiple outgoing messages and the implementation of the core operation. Consider a regular LDPC code of rate $R \geq 1 - j/k$, where j and k are the column and the row weight of its parity-check matrix, respectively. If this code is of high rate, then it consists of check nodes that are connected to many variables. For instance, the check nodes of LDPC codes with $R \geq 0.9$ and a column weight $j = 4$ are connected to more than 40 variables, i.e., $k > 40$. For an irregular LDPC code, the largest number of row weights can be even larger.

The core operation of the check-node update in (1) is the hyperbolic tangent function, which apparently seems difficult to implement in digital hardware. Analog realizations of the hyperbolic tangent function have been investigated in [8, 9], with the aim of facilitating extremely high-speed applications. The main difficulty in analog circuits is not the complexity of the hyperbolic tangent function but rather stability and synchronization issues. In Gallager's approach [1], the core operation of the check-node update in (1) was transformed into another form, i.e.,

$$2\tanh^{-1}(\tanh(\lambda/2)\tanh(\mu/2)) = \operatorname{sign}\lambda\operatorname{sign}\mu\cdot f(f(|\lambda|) + f(|\mu|)),$$

where

$$f(x) = \ln\frac{e^x + 1}{e^x - 1}$$

is an involution for $x > 0$. By introducing the involution $f(x)$, one can run check-node updates efficiently from both the computational- and time-complexity viewpoints in a way similar to the efficient implementation of symbol-node updates described above. However, it is not evident how to implement the involution function $f(x)$ efficiently in either analog or digital circuits because of its singularity at 0.

3. BINARY TREES FOR THE CHECK-NODE CONSTRAINTS

The motivation to consider binary trees is based on the fact that the complexity-intensive computation of the check-node update can be obtained by repeated application of the identity

$$2\tanh^{-1}(\tanh(\lambda/2)\tanh(\mu/2)) = \log\frac{1+e^{\lambda+\mu}}{e^{\lambda}+e^{\mu}}. \quad (2)$$

Moreover, as noted in [10], the right-hand side in (2) is the LLR of the binary sum of two statistically independent binary random variables U and V, where $\lambda = L(U)$ and $\mu = L(V)$ are their LLRs, i.e.,

$$L(U \oplus V) = \log\frac{1+e^{L(U)+L(V)}}{e^{L(U)}+e^{L(V)}}. \quad (3)$$

For the right-hand side in (3), which is a function of $L(U)$ and $L(V)$, we will use the well-established notation $L(U) \boxplus L(V)$ [10]. The domain of the $\boxplus$-operation can be extended to the set of real numbers together with $\pm\infty$. In this way, one obtains a monoid with ∞ as neutral element [10]. With this notation, the check-node update (1) can be rewritten as

$$L(r_{m-n}) = \boxplus_{n'\in\mathcal{L}(m)\setminus n} L(q_{n'-m}). \quad (4)$$

where $\boxplus_{n'=1\ldots k} L(U_{n'})$ denotes the k-fold $\boxplus$-sum.

Without loss of essential generality, we will focus on a single check node, say $m = 1$, and assume that it checks the binary input symbols $x_1, x_2, \ldots, x_k$. The factor graph for the check equation $x_1 \oplus \cdots \oplus x_k = 0$ is a tree with k leaves, in which the root corresponds to check node $m = 1$ and the k symbols represent the leaves. To reduce the check-node updates (1), operating on $k-1$ arguments, to a repeated application of the 2-argument formula (3), we transform the k-ary tree into a factor graph, which is a binary tree, by using suitable state variables S_ν. Two trees that achieve this task are shown in Fig. 1.

In Fig. 1, the double circles represent binary state variables. Each state variable, say S_ν, is the modulo-2 sum of the two symbol or state nodes, say U_ν and V_ν, checked by the binary check node leading towards state node S_ν (starting from the leaves towards the root), i.e., $S_\nu = U_\nu \oplus V_\nu$. The checl node updates (1) can be computed by making a forward and backward pass on either of these factor graphs using the coı function (3) or some approximation thereof [3, 4, 5]. For bot factor graphs in Fig. 1, the total computational load for chec node $m = 1$ consists of the forward recursive computation c $L(S_\nu)$ for the $k-2$ state variables, the backward recursion fc the latter, and the final backward recursion step to the leave which amounts to $3(k-2)$ core operations $L(U) \boxplus L($ (see [11] for the complexity computation on the maximall unbalanced tree).

The factor graph in Fig. 1(a) will be referred to as the ma imally unbalanced tree (note that this factor graph is topolo ically equivalent to a binary tree obtained by deleting all sta variables). Various simplifications of the SPA have been d rived from this maximally unbalanced tree [3, 5]. The oth factor graph, Fig. 1(b), which is a balanced tree, has bee proposed in [6] and implicitly in [4]. For the design of pa allel algorithms, the balanced tree clearly results in a muc smaller delay, i.e. about $2\log k$ core operations, compare with a delay of about k consecutive core operations for th maximally unbalanced tree (when running the forward an backward passes in parallel).

In this paper, we propose an algorithm that operates o *a strongly reduced factor graph for each single parity-checl* equation. This simplified balanced factor graph is obtainec by searching for the z ($z \geq 2$) least reliable symbol nodes ir each parity-check equation.

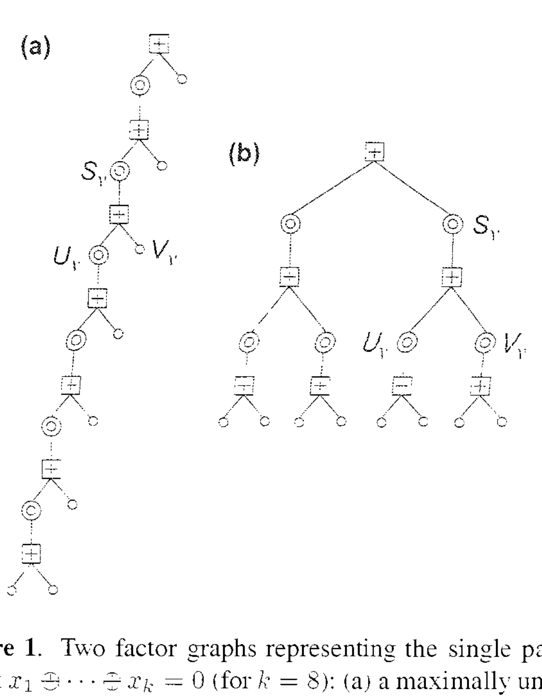

Figure 1. Two factor graphs representing the single parity check $x_1 \oplus \cdots \oplus x_k = 0$ (for $k = 8$): (a) a maximally unbalanced and (b) a balanced tree.

4. ORDERED STATISTICS FOR CHECK-NODE UPDATES

This section is devoted to simplifying the balanced tree factor graph by using the ordered statistics of the reliability values of the symbol nodes. An example of such a simplification is given by a parallel min-sum version of the SPA as described in the following subsection.

4.1 PARALLEL MIN-SUM CHECK-NODE UPDATES

We consider the SPA on the balanced tree with a simplified *core function based on the well-known minimum approximation [10]*

$$L(U) \boxplus L(V) \approx \operatorname{sign}(L(U))\operatorname{sign}(L(V)) \cdot \min\{|L(U)|, |L(V)|\}. \quad (5)$$

This simplified SPA is known as the "min-sum algorithm" [6]. By performing the min-sum update rule on the factor graph for the check equation $x_1 \oplus \ldots \oplus x_k = 0$ [see Fig. 1(b)], one obtains approximate values for the extrinsic LLRs $L(r_{1-n})$, $n = 1, \ldots, k$, which are given by

$$L(r_{1-n}) \approx \begin{cases} \text{sign}[L(S)]\text{sign}[L(u_n)]L_m & \\ \qquad \text{if } L_m < |L(u_n)| \\ \text{sign}[L(S)]\text{sign}[L(u_n)]L_{m'} & \\ \qquad \text{otherwise.} \end{cases} \tag{6}$$

Here $S = x_1 \boxplus \ldots \boxplus x_k$, and L_m and $L_{m'}$ denote the LLRs of the least and second least reliable symbol node, respectively. In particular, L_m and $L_{m'}$ are obtained by partially sorting the reliability magnitudes $|L(u_1)|, |L(u_2)|, \cdots, |L(u_k)|$ to obtain the two symbol nodes with the two smallest values.

The approximations (6) can be computed directly from $\text{sign}[L(S)]$, which is obtained from the forward pass, and no backward recursion is needed, provided that the LLRs L_m and $L_{m'}$ of the two least reliable symbol nodes are known. The computation of L_m and $L_{m'}$ is a partial-sorting problem, which can be done efficiently in $\log_2(k)$ time. Note that $\text{sign}[L(S)]$ is obtained by a simplified forward pass, which requires only sign computations. Furthermore, this forward pass and the partial sorting can be done in parallel on two trees of depth $\log_2(k)$. Hence, the min-sum algorithm for the check-node updates runs in time $\log_2(k)$, where the core operations are either comparisons or multiplications of two signs. The total number of operations required is about $k \log_2(k)$ comparisons and k sign operations.

4.2 SORTING-BASED CHECK-NODE UPDATES

By carrying the sorting idea of the preceding subsection further, we will obtain a family of reduced-complexity algorithms that approximate the check-node updates in the SPA. The balanced tree for the forward recursion can also be considered as a diagram for the sorting algorithm often referred to as merge-sorting [12] that is based on merging two ordered lists. Starting with the magnitudes $|L(x_1)|, |L(x_2)|, \cdots, |L(x_k)|$ at the leaves of the tree, these k nonnegative values can be ordered within a time complexity of about $\log_2(k)$ comparison operations. The sorting algorithm determines a permutation

$$\pi = \begin{pmatrix} 1 & 2 & \ldots & k \\ i_1 & i_2 & \ldots & i_k \end{pmatrix} \tag{7}$$

of the k symbols such that the permuted reliability values are ordered, i.e., $|L(x_{i_1})| \leq |L(x_{i_2})| \leq \cdots \leq |L(x_{i_k})|$.

The key idea of the new algorithm is to select a fixed number, say z, of least reliable symbol nodes $x_{i_1}, \ldots, x_{i_z}$, and to treat all remaining symbol nodes as being fully reliable, i.e.,

$$|L(x_{i_{z+1}})| = |L(x_{i_{z+2}})| = \cdots = |L(x_{i_k})| = \infty. \tag{8}$$

This amounts to keeping the original soft values for the z least reliable symbol nodes and approximating the original soft values for the other $k - z$ symbol nodes by the hard decisions $\pm\infty$.

The distinction of symbol nodes with soft and hard decisions can be reflected in the construction of a partially balanced binary factor graph for the updates of the check node

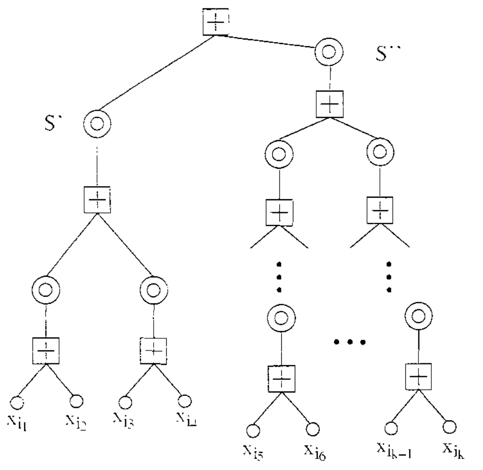

Figure 2. Reliability-based balanced tree separating the $z = 4$ least reliable and the $k - z$ most reliable symbol nodes in parity check $x_1 \boxplus \cdots \boxplus x_k = 0$.

corresponding to $x_1 \boxplus \ldots \boxplus x_k = 0$ (see Fig. 2). This partially balanced binary tree, which we will call a reliability-based balanced tree, is obtained as follows. From the soft value symbols $x_{i_1}, \ldots, x_{i_z}$, we form a balanced binary subtree with the root given by the binary state variable $S' = x_{i_1} \boxplus \ldots \boxplus x_{i_z}$. Similarly, a balanced binary sub-tree with the root $S'' = x_{i_{z+1}} \boxplus \ldots \boxplus x_{i_k}$ is formed from the hard-decision symbol nodes $x_{i_{z+1}}, \ldots, x_{i_k}$. Eventually, the two sub-trees are connected via a binary parity-check node, which becomes the root of the reliability-based balanced tree.

The complexity of the SPA on the partially balanced factor graph is mainly determined by the complexity on the left subtree emanating from state node S' because full soft reliability values are computed only on this sub-tree. On the right subtree, which emanates from state S'', the forward pass of the SPA needs only sign computations because all leaves have LLRs that are $\pm\infty$ (note that ∞ is the neutral element with respect to the $\boxplus$-operation).

To obtain a low-latency implementation of the check-node updates, we simplify the partially balanced tree of Fig. 2 further by keeping only the left sub-tree with the root S'. The binary value of the symbol node at the root, $S' = \sigma$, is determined by the hard-decision symbols, i.e.,

$$\begin{aligned} (-1)^\sigma &= \prod_{n=z+1}^{k} \text{sign}(L(x_{i_n})) \\ &= \text{sign}(L(S)) \prod_{n=1}^{z} \text{sign}(L(x_{i_n})). \end{aligned} \tag{9}$$

The resulting factor graph is shown in Fig. 3. Note that this graph can be further simplified by removing the state node S' and by incorporating the constraint $S' = \sigma$ into the check node at the root. The SPA on this simplified factor graph can either be carried out in its full version or by using some simplification for the $\boxplus$-operations (see e.g. [4]).

For $z = 2$, the algorithm essentially corresponds to the min-sum approximation. However, there is the following slight difference: in the min-sum algorithm the reliability of the messages for the hard-decision nodes are approximated

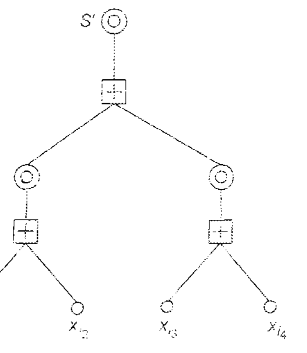

Figure 3. Balanced subtree of the $z = 4$ least reliable symbol nodes.

by the minimum $|L(x_{i_1})|$ in contrast to the approximation $|L(x_{i_1})| \boxplus |L(x_{i_2})|$ in the new algorithm.

For high-rate LDPC codes, the sorting-based check-node updates provide substantial savings in complexity: compared to the $3(k-2)$ $\boxplus$-operations in the full version, the simplified version needs only $3(z-2)+1$ $\boxplus$-operations, to which the small overhead for searching and for the sign operations has to be added.

Note that the partial sorting idea can also be applied to Gallager's decoding approach as described in Section 2. The reliability values for the outgoing messages are then computed from the z-fold $\boxplus$-sum $L(x_{i_1}) \boxplus L(x_{i_2}) \boxplus \ldots \boxplus L(x_{i_z})$.

4.3 FAST PARTIAL SORTING

The partial-sorting problem, which finds the z smallest elements in a set of k real values, is a problem whose worst-case complexity is difficult to analyze [12]. Here, we give a simple algorithm that provides a simple upper bound on the number of comparisons needed. The algorithm is based on merge-sorting as described above but with the following modification: in the ordered list obtained from the merging of two smaller ordered lists, only the z smallest values are kept and all other (larger) values are deleted from the list. As a result, the maximum list size at each stage of the algorithm is z. A balanced tree with k leaves contains a total of $k-1$ inner nodes (that are not leaves). At these inner nodes, at most z comparisons have to be done. Hence, the total complexity of partial sorting is upper-bounded by $z(k-1)$ comparisons.

Note that when the comparisons are done in parallel at each level in the tree, the run time of the algorithm corresponds to *about $\log_2(k)$ comparison operations.*

4.4 COMPLEXITY ANALYSIS

A complexity comparison of the full SPA and the proposed sorting-based (SB) version with z least reliable symbol nodes, the z-SB-SPA, will depend on the particular implementation of the core operations, which are the $\boxplus$-operation, the addition of real numbers $+$, the comparison of real numbers $<$, and the sign operation. The sign operation $(-1)^a(-1)^b = (-1)^{a+b}$ can be realized by the XOR-sum of the exponents; it is much less complex than the other three operations, which operate on real numbers, and, therefore, it will be neglected in the complexity analysis.

Let $T(\boxplus)$, $T(+)$ and $T(<)$ denote the time complexity in seconds of the three considered operations. In terms of run time, the three operations have the ranking

$$T(\boxplus) > T(+) > T(<)$$

and a similar ranking applies to the complexity of a hardware implementation, e.g., in terms of number of gates or chip area. Typically, $T(\boxplus)$ is much larger than $T(+)$ and $T(<)$.

The full SPA, the min-sum algorithm and the SB-SPA make identical symbol node updates but they differ in the way the check-node update are computed. Therefore, we will restrict the complexity analysis to the check-node update. In the sequel, we will consider a single check node, which is connected to k symbol nodes.

For the min-sum algorithm, we compute the minimum and second smallest value of the k incoming reliability magnitudes, which requires $2(k-1)$ comparisons when using the partial sorting algorithm described above. From these two least reliable values all the k outgoing messages can be determined. Thus, the total complexity amounts to $2(k-1)$ comparisons and the run time (in seconds) is $\lceil \log_2(k) \rceil\ T(<)$, where $\lceil c \rceil$ denotes the smallest integer larger or equal to c.

The check-node update for the full SPA is computed using the balanced factor graph as shown in Fig. 1(b), which results in a total complexity of $3(k-2)$ $\boxplus$-operations and a run time (in seconds) of $2\lceil \log_2(k) \rceil\ T(\boxplus)$.

The z-SB-SPA proceeds in two steps to compute the check-node updates. The first step consists of fast partial sorting, which allows to identify the z-least reliable symbol nodes. In the same step, one can also obtain all the sign-information (the complexity of which is negligible as explained above). Once the z least reliable symbol nodes have been identified, they can be grouped in a balanced subtree as shown in Fig. 3. In a second step, one runs the SPA on this subtree to obtain the outgoing messages that will be passed to the z least reliable symbol nodes. Parallel to this second step, the outgoing messages to all the other symbol nodes are computed. These messages all have the same magnitude, which is determined by the message that is passed to the state node S' as a result of the SPA on the balanced subtree. The total complexity of the z-SB-SPA amounts to $z(k-1)$ comparisons and $3(z-2)+1$ $\boxplus$-operations and the run time (in seconds) is $\lceil \log_2(k) \rceil\ T(<) + 2\lceil \log_2(z) \rceil\ T(\boxplus)$.

Table 1 contains a summary of the complexity computa*tions for the three considered algorithms. The simulation re*sults in the following section suggest that appropriate values for z are 3 and 4. It is evident from Table 1 that for high rate LDPC codes substantial savings in complexity and delay can be achieved with the $z = 3$ or 4 using the z-SB-SPA algorithm.

Example 1 *Consider the regular rate-$4158/4489$ LDPC code with column weight $j = 5$ and row weight $k = 67$, which is based on the array construction [13]. The updating of a check-node requires 195 $\boxplus$-operations, 132 $<$-operations, and 264 comparisons and 7 $\boxplus$-operations for the full SPA, the min-sum algorithm, and the z = 4-SB-SPA, resp., and the run times for the three algorithms are 14 $T(\boxplus)$, 7 $T(<)$, and 7 $T(<)$ + 4 $T(\boxplus)$, respectively.*

Table 1. Check-node update complexity for the full SPA, the min-sum and the z-SB-SPA.

algorithm	Full SPA	Min-Sum	z-SB-SPA
# operations (op)	$3(k-2)$ $\boxplus$-op	$2(k-1)$ $<$-op	$z(k-1)$ $<$-op and $3(z-2)-1$ $\boxplus$-op
run time	$2\lceil\log_2 k\rceil T(\boxplus)$	$\lceil\log_2 k\rceil T(<)$	$\lceil\log_2 k\rceil T(<)-2\lceil\log_2(z)\rceil T(\boxplus)$

If we assume that the $\boxplus$-operation is about 10 times as complex as the comparison operation (both in the number of gates and in terms of run time), then the z-SB-SPA achieves savings in complexity compared to the full SPA by a factor of about 5.8 and a savings in run time by a factor of almost 3. The savings of the min-sum algorithm over the full SPA are even larger, namely, by a factor of about 14.7 in complexity and a factor of 20 in run time, however, at the cost of a noticeable performance penalty.

5. SIMULATION RESULTS

For simulations on the binary-input AWGN channel, we have considered an LDPC code based on the array code construction of length $N = 4489$ and rate $4158/4489$, which is defined by $M = 335$ parity checks [13]. Fig. 4 shows the bit-error-rate performance of this code. The following LDPC decoding algorithms have been used: the full SPA, the SB-SPA with $z = 2, 3$, and 4 least reliable values. The results are obtained using Monte Carlo simulations, in which the maximum number of iterations is fixed to 80 in all cases. For the core operation $\boxplus$, we use Eq. (3) in its full accuracy.

We observe that the simple min-sum approximation, which essentially corresponds to the SB-SPA with only $z = 2$ least reliable values, suffers a performance penalty of about 0.3 dB at a bit-error rate of 10^{-6}. It is apparent from Fig. 4 that the loss in performance is recovered by increasing the number z of remaining least reliable values. For instance, for $z = 3$ the performance loss falls within 0.1 dB of the the full SPA, and for $z = 4$ it is only 0.04 dB from the full SPA.

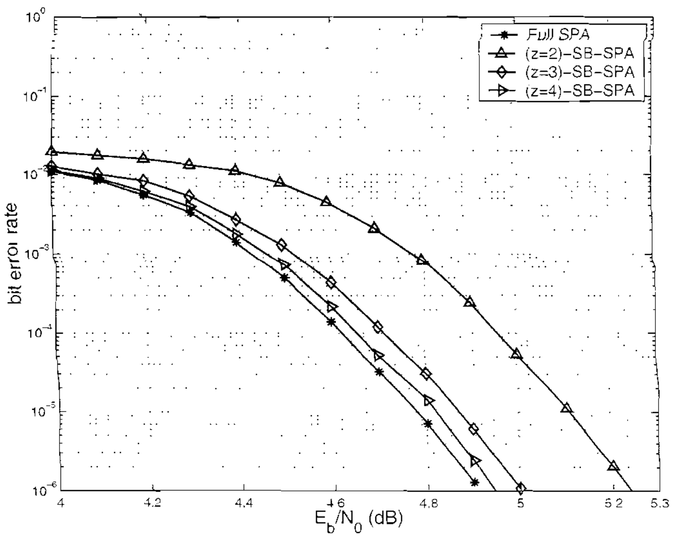

Figure 4. Performance of the SPA and its simplifications using $z = 2, 3$, and 4 least reliable values for a rate-4158/4489 code on the AWGN channel.

Fig. 5 shows the bit-error-rate performance of an LDPC code of length $N = 1008$ and rate 1/2 from the on-line repository [14] on the binary-input AWGN channel. Again we see that the simple min-sum approximation suffers nonnegligible performance loss relative to the full SPA, which can be regained by using $z = 3$ or $z = 4$.

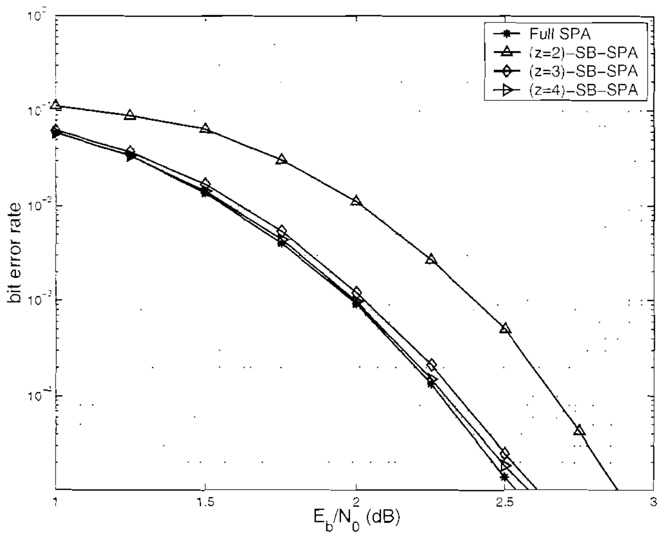

Figure 5. Performance of the SPA and its simplifications using $z = 2, 3$, and 4 least reliable values for a rate-1/2 code (1008, 504) on the AWGN channel.

6. CONCLUSIONS

A family of low-complexity and low-latency algorithms that approximate the sum–product algorithm has been proposed. The main feature of the simplification consists in ordering the reliability of the incoming message at each check node. The complexity of the simplified algorithm depends on the number z of least reliable messages that are selected. When keeping only $z = 2$ least reliable messages, the SB-SPA essentially reduces to the well-known min-sum algorithm, which has the least complexity. Simulation results have shown that for increasing values of z, the performance of the algorithm quickly approaches the performance of the full SPA. By suitably selecting the parameter z, the SB-SPA provides the flexibility to improve performance at the cost of increased complexity. Moreover, one can simplify the $\boxplus$-core operation by some reduced-complexity implementation of one's own choice.

7. ACKNOWLEDGMENTS

We thank Evangelos Eleftheriou and Sedat Ölçer for help-

ful discussions and comments on an early version of the paper.

REFERENCES

[1] R. G. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. Inform. Theory*, pp. 21–28, 1962.

[2] S.-Y. Chung, G. D. Forney, Jr., T. Richardson, and R. Urbanke, "On the Design of Low-Density Parity-Check Codes within 0.0045 dB of the Shannon Limit," *IEEE Commun. Lett.*, Vol. 5, No. 2, pp. 58–60, Feb. 2001.

[3] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced Complexity Iterative Decoding of Low–Density Parity–Check Codes Based on Belief Propagation," *IEEE Trans. Commun.*, vol. 47, pp. 673-680, May 1999.

[4] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient Implementations of the Sum-Product Algorithm for Decoding LDPC Codes," in *Proc. Globecom 2001*, San Antonio, TX, USA, pp. 1036A–1036E.

[5] E. Eleftheriou, T. Mittelholzer, and A. Dholakia, "A Reduced-Complexity Decoding Algorithm for Low-Density Parity-Check Codes," *IEE Electron. Lett.*, Vol. 37, pp. 102–104, Jan. 2001.

[6] F. R. Kschischang, B. J. Frey, H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Trans. Inform. Theory*, Vol. 47, No. 2, Feb. 2001, pp. 498–519.

[7] D. J. C. MacKay, "Good Error-Correcting Codes Based on Very Sparse Matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, Mar. 1999.

[8] J. Hagenauer, M. Moerz, and E. Offer, "Analog Turbo-Networks in VLSI: The Next Step in Turbo Decoding and Equalization," in *Proc. 2nd Intl. Symp. Turbo Codes & Related Topics*, Brest, France, Sept. 2000, pp. 209–218.

[9] H.-A. Loeliger, F. Lustenberger, and M. Helfenstein, and F. Tarkoy, "Probability Propagation and Decoding in Analog VLSI," *IEEE Trans. Inform. Theory*, vol. 47, pp. 837–843, Feb. 2001.

[10] J. Hagenauer, E. Offer, and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 429–445, Mar. 1996.

[11] S. M. Aji and R. J. McEliece, "The Generalized Distributive Law," *IEEE Trans. Inform. Theory*, Vol. 46, No. 2, pp. 325–343, Mar. 2000.

[12] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching*, Vol. 3, Addison-Wesley, 1973.

[13] J. L. Fan, "Array codes as Low-Density Parity-Check Codes," *Proc. 2nd Intl. Symp. on Turbo Codes*, Brest, France, Sept. 2000, pp. 543-546.

[14] David J.C. MacKay, Encyclopedia of Sparse Graph Codes `http://www.inference.phy.cam.ac.uk/mackay/codes/data.html`

Xiao-Yu Hu received the Ph.D. in computer and communication science from the Swiss Federal Institute of Technology Lausanne (EPFL) in 2002. Prior to that, he studied at the East China Institute of Technology and Southeast University, China. Since September 1999, he has been working at the IBM Zurich Research Laboratory, Rüschlikon, Switzerland, first as a Pre-Doc fellow, and now as a Post-Doc research fellow, on xDSL and magnetic recording projects. His research interests include information and communication theory, signal processing, channel coding, especially codes on graphs, and distributed/parallel algorithms.

Thomas Mittelholzer received a diploma in mathematics, the Ph.D., also in mathematics, and a diploma in computer science from the Swiss Federal Institute of Technology (ETH) Zurich, Switzerland, in 1981, 1987, and 1988, respectively. From 1989 to 1993 he was a Research Associate with the Signal and Information Processing Laboratory at the Swiss Federal Institute of Technology. He spent the year 1994 as Visiting Post-Graduate Research Scientist in the Center for Magnetic Recording Research at the University of California at San Diego. After three years as consultant in a small company in the security and crypto business, he joined the IBM Zurich Research Laboratory in 1999, where he works on magnetic recording. His current research interests include coding and communication theory, signal processing and coding for magnetic recording channels, and cryptography. Dr. Mittelholzer is a senior member of the IEEE Information Theory Society, and was member of the technical program committee for the 2002 International Telecommunications Symposium (ITS2002) in Natal, Brazil.