A Reduced Latency List Decoding Algorithm for Polar Codes

Jun Lin, Chenrong Xiong and Zhiyuan Yan
Department of Electrical and Computer Engineering, Lehigh University, PA, USA
Email: {jui311, chx310, yan}@lehigh.edu

Abstract—The cyclic redundancy check (CRC) aided successive cancelation list (SCL) decoding algorithm has better error performance than the successive cancelation (SC) decoding algorithm for short or moderate polar codes. However, the CRC aided SCL (CA-SCL) decoding algorithm still suffers from long decoding latency. In this paper, a reduced latency list decoding (RLLD) algorithm for polar codes is proposed. For the proposed RLLD algorithm, all rate-0 nodes and part of rate-1 nodes are decoded instantly without traversing the corresponding subtree. A list maximum-likelihood decoding (LMLD) algorithm is proposed to decode the maximum likelihood (ML) nodes and the remaining rate-1 nodes. Moreover, a simplified LMLD (SLMLD) algorithm is also proposed to reduce the computational complexity of the LMLD algorithm. Suppose a partial parallel list decoder architecture with list size \( L = 4 \) is used, for an \((8192, 4096)\) polar code, the proposed RLLD algorithm can reduce the number of decoding clock cycles and decoding latency by 6.97 and 6.77 times, respectively.

I. INTRODUCTION

Polar codes [1] are a significant breakthrough in coding theory, since it is proved that polar codes can achieve the channel capacity of binary-input symmetric memoryless channels in [1] and any discrete or continuous channel in [2]. Polar codes can be efficiently decoded by the low-complexity successive cancelation (SC) decoding algorithm [1] with complexity of \( O(N \log N) \), where \( N \) is the block length.

Lots of efforts [3], [4] have already been devoted to improve the error-correction performance of polar codes with short or moderate lengths. An successive cancelation list (SCL) decoding algorithm was recently proposed in [3], performs better than the SC decoding algorithm and performs almost the same as a maximum-likelihood (ML) decoder [3]. In [4], the cyclic redundancy check (CRC) is used to pick the output codeword from \( L \) candidates, where \( L \) is the list size. The CRC-aided SCL decoding algorithm performs much better than the SCL decoding algorithm at the expense of negligible loss in code rate. For example, it was shown in [4] that the CRC-aided SCL decoding algorithm outperforms the SC decoding algorithm by more than 1 dB when the bit error rate (BER) is on the order of \( 10^{-5} \) for a polar code of length 2048.

Many research efforts [5]–[9] have been devoted to the reduction of the decoding latency of the SC decoding algorithm. The simplified successive cancelation (SSC) and the ML-SSC decoding algorithms were proposed in [5] and [7], respectively. Both SSC and ML-SSC decoding algorithms can reduce the decoding latency of a SC decoder significantly. However, the reduced latency list decoding algorithm has been rarely discussed in open literature.

In this paper, the algorithms that reduce the latency of list polar decoders are investigated. The main contributions are shown as follows.

1) A reduced latency list decoding (RLLD) algorithm over LLR domain for polar codes is proposed. The proposed RLLD algorithm deals with rate-0 nodes and part of rate-1 nodes in the same way as the SSC decoding algorithm.

2) A list ML decoding (LMLD) algorithm is proposed to decode the ML and remaining rate-1 nodes. For the list size \( L < 8 \), a hardware friendly simplified LMLD (SLMLD) algorithm is also proposed.

3) For list size \( L = 4 \), an efficient hardware architecture for the proposed SLMLD algorithm is presented. Under a TSMC 90nm technology, at the cost of 1.07 million standard NAND gates, the proposed architecture can achieve a frequency of 400MHz with 4 stage of pipelines.

4) For a partial parallel decoder architecture with \( L = 4 \), it is shown that the RLLD with the SLMLD algorithms can reduce the decoding cycles and latency by 6.97 and 6.77 times, respectively.

II. PRELIMINARIES

A. Polar codes encoding

The generation matrix of a polar code is an \( N \times N \) matrix \( \mathbf{G} = \mathbf{B}_N \otimes \mathbf{F}_N^{\otimes n} \), where \( N = 2^n \), \( \mathbf{B}_N \) is the bit reversal permutation matrix, and \( \mathbf{F}_N = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \). Here \( \otimes \) denotes the \( n \)-th Kronecker power and \( \mathbf{F}_N \otimes \mathbf{F}_N = \mathbf{F}_N \otimes \mathbf{F}_N^{(n-1)} \). Let \( \mathbf{u}_N^{N-1} = (u_0, u_1, \ldots, u_{N-1}) \) denote the data bit sequence and \( \mathbf{x}_0^{N-1} = (x_0, x_1, \ldots, x_{N-1}) \) the corresponding encoded bit sequence, then \( \mathbf{x}_N^{N-1} = \mathbf{u}_N^{N-1} \mathbf{G} \). The indices of the encoding bit sequence \( \mathbf{u}_N^{N-1} \) are divided into two sets: the information bits set \( \mathcal{A} \) contains \( K \) indices and the frozen bits set \( \mathcal{A}^c \) contains \( N - K \) indices. \( \mathbf{u}_A \) are the information bits whose indices all come from \( \mathcal{A} \). \( \mathbf{u}_{\mathcal{A}^c} \) are the frozen bits whose indices come from \( \mathcal{A}^c \). The encoding graph of a polar code with \( N = 8 \) is shown in Fig. 1.

B. SSC and ML-SSC Decoding Algorithms

A polar code of length \( N = 2^n \) can also be represented by a full binary tree of depth \( n \) [5], where each node of the tree is associated with a constituent code. The binary tree representation of an \((8, 3)\) polar code is shown in Fig. 2, where the black and white leaf nodes correspond to information and frozen bits, respectively. In order to show the connection between the tree representation and the direct encoding graph...
in Fig. 1, the constituent code associated with each tree node is also shown in Fig. 2. There are three types of nodes in a binary tree representation of a polar code: rate-0, rate-1 and arbitrary rate nodes. The leaf nodes of a rate-0 and rate-1 nodes are associated with only frozen and information bits, respectively. The leaf nodes of an arbitrary rate node are associated with both information and frozen bits. For example, the rate-0, rate-1 and arbitrary rate nodes in Fig. 2 are represented by circles in white, black and gray, respectively.

The SC decoding algorithm can also be mapped on a binary tree, where each node acts as a decoder for its constituent code. As shown in Fig. 2, the decoder at node $v$ receives a soft information vector $\alpha_v$ and returns its corresponding constituent code $\beta_v$. The SC decoding algorithm is initialized by feeding the root node with the channel LLRs, $(L_0, L_1, \ldots, L_{N-1})$, where $L_0 = \log(Pr(y_i|x_i = 0)/Pr(y_i|x_i = 1))$. When an internal node $v$ is activated, it calculates the soft information vector $\alpha_v$ sending to its left child, where

$$\alpha_v[i] = f(\alpha_v[2i], \alpha_v[2i+1]) \quad \text{for} \quad 0 \leq i < 2^{n-t},$$

$$f(a, b) = 2 \tanh^{-1}(\tanh(a/2) \tanh(b/2)), \quad \text{and} \quad t \quad \text{is the layer index of the child node}.$$.

Node $v$ then waits until it receives the constituent code $\beta_v$. The soft information vector $\alpha_v[i] = \alpha_v[2i](1 - 2\beta_v[i]) + \alpha_v[2i + 1]]$ for $0 \leq i < 2^{n-t}$. (3)

Once the right child returns its constituent code $\beta_v$, node $v$ computes its constituent code $\beta_v$ as:

$$(\beta_v[2i], \beta_v[2i + 1]) = (\beta_v[i] \oplus \beta_v[i], \beta_v[i]),$$

where $0 \leq i < 2^{n-t}$ and $\oplus$ is modulo-2 addition. When a leaf node $v$ is activated, its constituent code $\beta_v$ is set to 0 if leaf node $v$ is associated with a frozen bit. Otherwise, $\beta_v$ is calculated from $\alpha_v$ with the threshold detection:

$$\beta_v = \begin{cases} 0 & \alpha_v \geq 0 \\ 1 & \alpha_v < 0 \end{cases}$$

(5)

From the root node, all nodes in a tree are activated in a recursive way for the SC decoding. Once $\beta_v$ for the last leaf node is generated, the codeword $x_0^{N-1}$ can be obtained by combining and propagating $\beta_v$ up to the root node.

The SSC decoding algorithm in [5] simplifies the decoding of rate-0 and rate-1 nodes. Once a rate-0 node is activated, it immediately returns its constituent code which is an all zero vector. Once a rate-1 node is activated, its constituent code is directly calculated from the received soft information vector with the threshold detection rule shown in Eq. (5). The ML-SSC decoding algorithm [7] simplifies the SSC decoding algorithm further by performing the exhaustive-search ML decoding on some resource constrained arbitrary rate nodes, which are called ML nodes in [7]. For an ML node with layer index $t$, the associated constituent code is estimated according to:

$$\beta_v = \arg \max_{\alpha \in \mathbb{C}} \sum_{i=0}^{2^{n-t}-1} (1 - 2\alpha[i])\alpha_v[i],$$

(6)

where $\mathbb{C}$ is the set of possible constituent codes for the ML node. The binary tree representations of the example (8, 3) polar code under SSC and ML-SSC decoding algorithms are shown in Fig. 3 (a) and (b), respectively. It is observed that the SSC decoding algorithm can reduce the number of nodes to be activated. This number is further reduced by applying the ML-SSC decoding algorithm which introduces ML nodes. It is obvious that all the child nodes of a rate-0 and rate-1 node are still rate-0 and rate-1 nodes, respectively. During the reduction of the binary tree, a rate-0 or rate-1 node is kept only if their parent nodes are not a rate-0 or rate-1 node, respectively. For an arbitrary rate node $v$, let $n_v$ and $d_v$ denote the number of leaf nodes and the number of leaf nodes that correspond to information bits, respectively. In [7], an arbitrary rate node is labeled as an ML node only if its $n_v$ and $d_v$ do not exceed predefined values.

![Binary tree representations of a (8, 3) polar code under SSC and ML-SSC decoding algorithms](image)

**C. LLR Based List Decoding Algorithms**

In the first several works [3], [10], [11] on list decoding of polar codes, the list decoding algorithm is performed either on probability or logarithmic likelihood (LL) domain. In [12], an LLR based list decoding algorithm is proposed to reduce the message memory requirement and the computational complexity of LL based list decoding algorithm. The LLR based
list decoding algorithm employs a novel path metric \( \text{PM}_i^{(j)} \), which is computed as:

\[
\text{PM}_i^{(j)} = \sum_{k=0}^{L} m_i|\mathbf{L}(k)[i]|,
\]

where \( m_i = 1 \) only if \( \hat{u}_i[i] = \delta(\mathbf{L}_n^{(k)}[i]) \) and \( \delta(x) = \frac{1}{2}(1 - \text{sign}(x)) \) [12]. Otherwise \( m_i = 0 \). Here \( \mathbf{L}_n^{(k)}[i] \) denotes the number of reliable decoding paths in \( \mathbf{L}_n^{(k)} \) and \( y \) is the received channel soft information vector.

### III. THE PROPOSED RLLD ALGORITHM

Though existing list decoding algorithms for polar codes can improve the performance of SC decoders significantly. They still suffer from long decoding latency. During the decoding of each information bit, the current decoding paths need to be doubled and at most \( L \) most reliable decoding paths are kept, where \( L \) is the list size. The extra cycles spent on path pruning increase the number of the overall decoding cycles [10]. In this paper, a reduced latency list decoding (RLLD) algorithm for polar codes is proposed. Let \( W_\text{e} \) and \( \text{I}_\text{e} \) denote the number of leaf nodes and leaf nodes associated with information bits of a node \( v \) in a binary tree, respectively. Let \( W_\text{T} \) be a predefined threshold value. The general architecture of the proposed RLLD algorithm is shown as follows:

1) For a binary tree representation of a polar code, label all the rate-0, rate-1 and ML nodes. For a node \( v \) in the tree, let \( W_\text{e} \) and \( \text{I}_\text{e} \) denote the numbers of reliable decoding paths and leaf nodes associated with information bits, respectively. For rate-1 nodes, \( \text{I}_\text{e} = W_\text{e} \). Moreover, two types of nodes are defined: \( \text{T}_0 \) and \( \text{T}_1 \). \( \text{T}_0 \) nodes do not include rate-1 nodes with \( \text{I}_v > W_\text{T} \) and all rate-0 nodes. \( \text{T}_1 \) nodes include rate-1 nodes with \( \text{I}_v \leq W_\text{T} \) and all ML nodes. For all ML nodes, \( W_\text{e} \leq W_{\text{ML}} \) and \( \text{I}_\text{e} \leq 8 \), where \( W_{\text{ML}} \) is also a predefined threshold value.

2) For each decoding path, perform the SC decoding algorithm on the corresponding pruned binary tree, if a \( \text{T}_0 \) node is activated, the corresponding constituent code is decoded immediately and sent to its parent node. Besides, it is unnecessary to compute the LLR vector sent to a rate-0 node, since the constituent code of a rate-0 node is always a zero vector.

3) If a \( \text{T}_1 \) node is activated, compute \( 2^{L_0} \) path metrics for each current decoding path, where each path metric corresponds to the reliability of a possible decoding path. Find at most \( L \) most reliable decoding paths and continue their corresponding SC decoding. Since only rate-1 nodes with \( \text{I}_v < W_\text{T} \) are involved in the list decoding, the choice of \( W_\text{T} \) should be decided by the numerical simulation.

4) Once all \( \text{T}_0 \) and \( \text{T}_1 \) nodes have been activated and all the SC decoding procedures on each decoding path are finished, perform cyclic redundancy check (CRC) on the information bits of each candidate codeword. The output codeword is the one that passes the CRC.

In terms of software or hardware implementation, the proposed RLLD algorithm can be performed over \( L \) LLR matrices and \( I \) bit matrices. For \( l = 0, 1, \ldots, L-1 \) and \( t = 1, 2, \ldots, n \), let \( P_{l,t} \) be a probability message array of \( 2^{n-t} \) elements: \( P_{l,t}[j] \) stores an LLR message for \( j = 0, 1, \ldots, 2^{n-t} - 1 \). The received channel LLRs are stored in \( P_{0,0} \) which has \( N = 2^n \) elements. \( C_{l,t} \) has a similar structure as \( P_{l,t} \). \( C_{l,t}[j] \) stores two binary partial sums \( C_{l,t}[1][0] \) and \( C_{l,t}[1][1] \) for \( j = 0, 1, \ldots, 2^{n-1}-1 \). Let \( r_l = (r_l[n-1], r_l[n-2], \ldots, r_l[0]) \) be the message updating reference index array for decoding path \( l \). For decoding path \( l \), \( r_l[0] = 0 \), while all other elements are initialized with 0. When a \( T_0 \) or \( T_1 \) node \( v \) is activated, the computation of the soft information vector sent to node \( v \) for decoding path \( l \) is shown in Algorithm 1, where \( t_v \) is the layer index of node \( v \) and \( P_{l,t_v} \) is the LLR vector sent to node \( v \). The \( g \) function is shown in Eq. (3). If node \( v \) is a rate-0 node, as mentioned before, it is unnecessary to compute the received LLR vector. Under this circumstance, \( t_v \) is decreased by 1. When a decoding path \( l \) needs to be copied to decoding path \( l' \), the lazy copy approach in [10] is applied. Instead of copying LLR matrices, \( r_l[I_l-1], \ldots, r_l[1] \) are copied to \( r_{l'}[I_l-1], \ldots, r_{l'}[1] \), respectively, while \( r_{l'}[0], \ldots, r_{l'}[I_l] \) are set to 0.

For decoding path \( l \), during the computation of \( P_{l,t} \), LLR arrays, \( P_{l,t}, \ldots, P_{l,t} \), need to be updated in serial, where \( \text{I}_v \) is a pre-computed layer index. For the tree representation of a polar code, suppose all leaf nodes from left to right are indexed from 0 to \( N-1 \). Let the indices of the leftmost and rightmost leaf nodes of the subtree of node \( v \) be \( \text{IDX}_v \) and \( \text{IDX}_l \), respectively. \( \text{I}_v \) is computed based on \( \text{IDX}_v \) as shown in Algorithm 2, where the function dec2bin computes the binary representation of its input and \( B_{n-1} \) and \( B_0 \) are the most and least significant bits, respectively.

---

**Algorithm 1:** \text{llrComp}(\ell, \alpha)

\text{input} : \ell, v_t
\text{output} : P_{\ell,t_\ell}

\begin{enumerate}
\item for \( \ell = \ell_s \) to \( v_t \) do
\item for \( k = 0 \) to \( 2^{n-\ell} \) do
\item if \( \ell = \ell_0 \) then
\item \( b_k = C_{\ell,t_\ell}[0] \)
\item \( P_{\ell,t_\ell}[k] = g(P_{\ell,t_\ell}[2^k], P_{\ell,t_\ell}[2^k+1], b_k) \)
\item else
\item \( P_{\ell,t_\ell}[k] = f(P_{\ell,t_\ell}[2^k], P_{\ell,t_\ell}[2^k+1]) \)
\end{enumerate}

---

**Algorithm 2:**

\text{input} : \text{IDX}_0
\text{output} : \ell_s

\begin{enumerate}
\item if \( \text{IDX}_0 == 0 \) then \( \ell_s = 0 \)
\item else
\item \( \ell_s = n \)
\item \( (B_n, B_{n-1}, \ldots, B_0) = \text{dec2bin}(\text{IDX}_0) \)
\item for \( j = 0 \) to \( n - 1 \) do
\item if \( B_j == 0 \) then \( \ell_s = \ell_s - 1 \)
\item else break
\end{enumerate}
Once the constituent code $C_i^v$ sent from node $v$ for decoding path $l$ is computed, $C_i^v$ is stored in $C_{l,v}[k][0]$ for $k = 0, 1, \ldots, 2^{n_l-1}$ if node $v$ is the left child of its parent node. Otherwise $C_i^v$ is stored in $C_{l,v}[k][1]$ for $k = 0, 1, \ldots, 2^{n_l-1}$. If the contents of decoding path $l$ need to be copied to decoding path $l'$, the partial sums in decoding path $l$ are copied to the corresponding locations in decoding path $l'$. If node $v$ is the right child of its parent node, then the partial sum computation for path $l$ is performed as shown in Algorithm 3. The input $I_v$ is a layer index and can be obtained by applying Algorithm 2 with IDX0 and $I_v$ being replaced with IDX1 and $I_v$, respectively.

**Algorithm 3: pSumComp(l, α)**

- **Input**: $I_v, t_0$
- **for** $t = t_0$ to $I_v$ **do**
  - **if** $t = I_v$ **then**
  - $C_{l,v}[2k][0] = C_{l,t-1}[k][0] \oplus C_{l,t-1}[k][1]$
  - $C_{l,v}[2k+1][0] = C_{l,t-1}[k][1]$
  - **else**
  - $C_{l,v}[2k][1] = C_{l,t-1}[k][0] \oplus C_{l,t-1}[k][1]$
  - $C_{l,v}[2k+1][1] = C_{l,t-1}[k][1]$


**A. LMLD Algorithms**

When a $T_1$ node is activated, the current decoding paths will expand, and at most $L$ most reliable decoding paths are kept. In this paper, a list ML decoding (LMLD) algorithm is proposed to find at most $L$ most reliable decoding paths. For a $T_1$ node $v$, there are $2^{n_v}$ candidate output constituent codes since the number of information bits associated with the leaf nodes of a node $v$ is $I_v$. Therefore, for each decoding path $l$, the proposed LMLD algorithm computes $2^{n_v}$ extended path metrics $PM^j_l$ for $j = 0, 1, \ldots, 2^{n_v} - 1$ based on the current path metric $PM_l^j$. Finding the $L$ most reliable surviving decoding paths is equivalent to finding the $L$ most reliable constituent codes among all candidates. Here, several conclusions are made on path metrics and extended path metrics:

- For each decoding path $l$, the path metric $PM_l$ is initialized with 0. The extended path metrics are computed only when a $T_1$ node is activated.

- For each decoding path $l$, each extended path metric $PM^j_l$ corresponds to the reliability measure of the associated candidate constituent code $C_{v,l}$ sent from node $v$.

- The extended path metric $PM^j_l$ is computed as shown in Eq. (8), where $NM^j_l$ is called node metric and $NM^j_l = \sum_{k=0}^{2^{n_v}-1} m_k |\alpha_{v,l}[k]|$. $\alpha_{v,l}$ is the LLR vector received by the node $v$. $m_k = 1$ only if $C_{v,l}[k] = \delta(\alpha_{v,l}[k])$, where $\delta(x) = \frac{1}{2}(1 - \text{sign}(x))$. Otherwise, $m_k = 0$. As shown in Eq. (8), for $k = 0, 1, \ldots, 2^{n_v} - 1$, if $C_{v,l}[k]$ does not equal to the threshold detection based on $\alpha_{v,l}[k]$, then $PM^j_l$ is punished by adding the absolute value of $\alpha_{v,l}[k]$. As a result, the smaller a extended path metric is, the more reliable a corresponding constituent code is.

$$PM^j_l = PM_l + NM^j_l$$ (8)

Based on the previous conclusions, the proposed LMLD algorithm finds the $L$ most reliable constituent codes by sorting the $L$ minimum metrics among $2^{n_v}L$ metrics. Let set $S = \{(l,j)|r = 0, 1, \ldots, L - 1\}$, where $(l,j)_r$ is the index of a candidate constituent code. Thus, the proposed LMLD algorithm is shown in Eq. (9),

$$S = \text{argmin}_{L} - {\sum_{l=0}^{L-1} PM^j_l}_j$$ (9)

where $\text{argmin}_{L} - \sum_{l=0}^{L-1} PM^j_l$ finds the associated indices of the $L$ minimum metrics among all input metrics. The current $L$ path metrics are updated with the $L$ minimum extended path metrics.

As shown in Eq. (9), the computational complexity of the proposed LMLD algorithm is exponential to $I_v$, which is the number of leaf nodes associated with information bits for node $v$. As a result, the maximum value of $I_v$ should be limited for practical implementation of the proposed LMLD algorithm. In this paper, the maximum value of $I_v$ is set to 8. The maximum number of leaf nodes of a ML node is set to $W_{ML} = 16$. In case of $W_T$ is greater than 8, the corresponding rate-1 node is split to several rate-1 nodes with $W_v = 8$. The other generated nodes due to the split are viewed as arbitrary rate nodes. Take a rate-1 node with $W_v = 32$ as an example, the split is shown in Fig. 4, where 4 rate-1 nodes with $W_v = 8$ are generated while the other generated nodes are deemed as arbitrary rate nodes. Besides, $W_v$ for a rate-1 node can only be a power of 2.

![Fig. 4. The tree split of a rate-1 node with $W_v = 32$](image)

**B. SLMLD Algorithms**

The computational complexity of the proposed LMLD algorithm is still high when $I_v$ is close to 8. In this paper, for $L \leq 8$, a simplified list ML decoding (SLMLD) algorithm suitable for parallel hardware implementation is proposed to reduce the computational complexity of the proposed LMLD algorithm in further. Here, $L$ is assumed to be a power of 2. The proposed SLMLD algorithm shown in Eq. (9) is divided into two major steps:

1) For each current decoding path $l$, find its most reliable $L$ constituent codes based on node metrics. Since only the $L$ most reliable constituent codes are needed at last and at most $L$ constituent codes are from the same decoding path $l$, it is enough to find the $L$ most reliable constituent codes for a decoding path $l$.

2) Compute the extended path metrics based on survived node metrics from previous step, and find the final $L$
most reliable constituent codes based on these $L \times L$ extended
extended metrics.

Depending on the value of $I_v$, the first step can be simplified
further. If $2^{L_v} \leq L$, nothing needs to be done. If $2^{L_v} = 2L$, the
minimum $L$ extended path metrics and their corresponding
$l$ and $j$ indices are computed with a bitonic sequence [13]
based sorter (BBS) [11], where the BBS first transforms
the inputs into a bitonic sequence and then generates $L$ minimum
metrics among all inputs. When $2^{L_v} > 2L$, the minimum $L$
node metrics are computed as follows:

- The $2^{L_v}$ node metrics are divided into $L$ groups as follows:

$$ \text{group 1} \quad n_1^0, \ldots, n_{L-1}^0, \ldots, n_{L-1}^{L_q-1} $$

$$ \text{group L} \quad n_1^{(L-1)q}, \ldots, n_{L-1}^{(L-1)q} $$

where $q = 2^{L_v} / L$. The minimum two metrics of each

group are then computed.

- Among the resulting $2L$ extended path metrics, the

minimum $L$ extended path metrics and their corre-

sponding $l$ and $j$ indices are computed with a BBS.

When list size $L = 2$, for any $I_v$ values, the first step is

just finding the minimum two extended path metrics and their

corresponding index pairs $(i, j)$'s.

The second step of the proposed SLMLD algorithm em-

ploys the $2L$-L BBS sorter with $2L$ inputs and $L$ outputs

repeatedly to generate $L$ final extended path metrics and their

associated path indices. Take $L = 4$ as an example, there are

$4L$ extended path metrics: $\text{PM}_0^0$, $\text{PM}_0^1$, $\ldots$, $\text{PM}_{4L-1}^0$, then

$\text{PM}_0^L$, $\ldots$, $\text{PM}_{2L-1}^L$ and $\text{PM}_{2L}^{L^2-1}$, $\ldots$, $\text{PM}_{4L-1}^{L^2-1}$ are applied
to two $2L$-L BBSs, respectively. Thus, total $2L$ metrics are

selected out. Then the $2L$-L BBS is employed again to
generate the final $L$ minimum extended path metrics: $\text{PM}_0^0$, $\ldots$, $\text{PM}_0^{L-1}$, $\ldots$, $\text{PM}_{L-1}^{L-1}$.

C. Simulation Results

For an (8192, 4096) polar code, the frame error rate (FER)

performance of the proposed RLLD algorithm is shown in

Fig. 5, under the AWGN channel with BPSK modulation.

As shown in Fig. 5, CSI denotes the CRC aided SC list
decoding algorithm [3] with list size $L = i$ over LLR domain,

and RS$(i, \omega)$ denotes the proposed RLLD algorithm with the

SLMLD algorithm when list size $L = i$ and $W_T = \omega$. For both

CSI and RS$(i, \omega)$ algorithms, 32 information bits are replaced

with a 32-bit CRC checksum.

For simplicity, the FER performances of the proposed

RLLD algorithm with LMLD (RL) algorithm are not shown

in this paper, since the FER performances of the RL algorithm

are the same as that of the CS algorithm with the same list

size.

Based on the simulation results, the following conclusions

are made:

- The performance of the proposed RS algorithm is

affected by the list size $L$. For the (8192, 4096) polar

code, the FER performances of RS(2, 8) is close

that of CS2. However, RS(4, 8) and RS(8, 8) show

performance degradation when the FER is blow $10^{-4}$.

- In order to achieve good error correction performance,

for the proposed RS algorithm, the threshold value

$W_T$ should be large enough. A larger $W_T$ will transfer

more rate-1 nodes to $T_i$ nodes, which in turn increases the

chance that a correct codeword shows in the final

lists. For the (8192, 4096) polar codes, RS(4, 8) and

RS(8, 8) perform worse than RS(4, 32) and RS(8, 32),

respectively, when the SNR is large.

- The side effect of increasing $W_T$ is that both the
decoding complexity and latency will increase since

more $T_i$ nodes are generated. Based on simulation

results shown in Fig. 5, a dynamic $W_T$ can be adopt

for the proposed RS algorithm in order to achieve the

most latency reduction at different SNR regions while

maintaining the error correction performance.

D. Hardware implementation of the proposed SLMLD

In this paper, an efficient hardware implementation of the

proposed SLMLD algorithm is shown in Fig. 6, where the

corresponding list size $L = 4$, and the architectures for other
$L$ values can be inferred. As shown in Fig. 6, the node metric
generation (NMG) unit finds $L$ minimum node metrics and their
corresponding constitution codes for each decoding path.

For the decoding path $l$, the extended path metrics $PM_l^j$’s are

obtained by adding the node metrics with the path metric $PM_l$,

which is stored in registers and initialized with 0. BBSs are

in Fig. 6 denotes the BBS with 8 metrics to be sorted. Two stages

of BBSs find the 4 minimum extended path metrics and their

corresponding constitution codes and list indices.

The hardware architecture of the NMG unit is shown in

Fig. 7. Since the maximum value of $I_v$ is 8 for any $T_i$ node,

there are at most $2^8 = 256$ candidate constituent codes for a $T_i$
node $v$. Each Enc unit in Fig. 7 is responsible for generating a

candidate constituent code based on the encoding of polar
codes. For $j = 0, 1, \ldots, 2^L - 1$, the LLR selection unit, $LS_j$, and

the summation unit, $SUM_j$, work together to compute the
node metric $N_{Mj}^{l}$ shown in Eq. (8). Based on the input LLR vector $\alpha_{\ell,l}$, LS$_j$ outputs an LLR vector which has the same amount of elements as that of $\alpha_{\ell,l}$. For $k = 0, 1, \cdots, 2^{n-1} - 1$, the $k$-th output LLR is 0 only if $m_k = 0$. Otherwise, the output LLR is $|\alpha_{\ell,l}[k]|$. The SUM$_j$ unit just adds up all its input LLRs sent from LS$_j$ and outputs the corresponding node metric. The minimum two LLRs computation (MC) unit in Fig. 7 finds the first and the second minimum LLRs and their corresponding constituent codes among all its inputs. When $L = 4$, as shown in Fig. 7, the computed node metrics are divided into 4 groups and fed to 4 MC units, respectively. The BBS$_{l+4}$ unit generates 4 finally survived node metrics and their corresponding constituent codes.

![Diagram](image-url)

Fig. 7. Hardware architecture of the proposed NMG unit

In this paper, the proposed architecture for the SLMLD algorithm is synthesized under a TSMC 90nm CMOS technology. With 4 stages of pipeline registers, it achieves a frequency of 400MHz and consumes 1.07 millisecond standard NAND gates. For our implementation, when a $T_1$ node is activated, it will take 4 clock cycles to find the surviving constituent codes and decoding paths. The area of the architecture of the SLMLD algorithm is almost the same as an LLR based list decoder with $L = 4$.

E. Comparisons of decoding clock cycles and latency

Since the detailed decoding cycles of list decoders are related with a detailed hardware architecture, in this paper, the decoding latency comparison is performed based on the assumption that the partial parallel list architecture [10] is employed and there are $P = 128$ processing units for each decoding path. Let $N_D$ be the clock cycles used to decode a codeword for decoders with the proposed RS algorithm. Then $N_D = N_{R} + N_{P}$, where $N_R$ and $N_P$ are cycles used on the LLR computation and path pruning, respectively. Besides, $N_P = N_xN_x$, where $N_x$ is the times that a $T_1$ node is activated and $N_x$ is the number pipelines inserted in the implementation of the SLMLD algorithm. Let $N_C$ denote the clock cycles used to decode a codeword for decoders with the CS algorithm. Then $N_C = 2N + \frac{N}{R}\log_2\left(\frac{\lambda}{D_N}\right) + NR$ [10], where $N$ and $R$ are the code block length and rate, respectively.

For the aforementioned (8192, 4096) polar code used in our simulations in Section III-C, $N_L = 1207$, $N_x = 441$ and $N_x = 4$ when $W_T = 32$ and $L = 4$. Thus, $N_D = 1207 + 441\times 4 = 2971$. Meanwhile, the cycles $N_C = 2\times 8192 + 8192 + \frac{8192}{125}\times \log_2\left(\frac{\lambda}{D_N}\right) + 4096 = 20736$. With the proposed RS decoding algorithm, the clock cycles used for decoding one codeword is reduced by about 6.97 times.

Under the UMC 90nm CMOS technology, the (8192, 4096) list polar decoder can achieve a frequency of 412MHz [12] when list size $L = 4$. Since the list decoder with the proposed RS decoding algorithm need only to change the path pruning part, the proposed list decoder can only achieve a frequency of 400MHz under 90nm technology. Thus, the decoding latency is reduced by about 6.77 times due to the proposed RS decoding algorithm when $L = 4$.

IV. CONCLUSION

In this paper, a reduced latency decoding algorithm for polar codes is proposed. The hardware implementation of the SLMLD is also discussed. The future work includes studying the performances of the proposed RLLD algorithm when FER is below $10^{-10}$. Besides, more efficient implementations of the proposed SLMLD algorithm when list size is large will be investigated.

REFERENCES


