

Stall Pattern Avoidance in Polynomial Product Codes

Carlo Condo, François Leduc-Primeau, Gabi Sarkis, Pascal Giard, *Member, IEEE*,
and Warren J. Gross, *Senior Member, IEEE*

Department of Electrical and Computer Engineering, McGill University, Montreal, QC, Canada

Abstract—Product codes are a concatenated error-correction scheme that has been often considered for applications requiring very low bit-error rates, which demand that the error floor be decreased as much as possible. In this work, we consider product codes constructed from polynomial algebraic codes, and propose a novel low-complexity post-processing technique that is able to improve the error-correction performance by orders of magnitude. We provide lower bounds for the error rate achievable under post processing, and present simulation results indicating that these bounds are tight.

I. INTRODUCTION

Product codes [1] are concatenated codes often considered for applications requiring high throughput and very low bit error rate (BER) [2], [3]. They allow efficient construction of long codes from short component codes. The concatenated structure and code length guarantee very good error-correction performance, while low decoding latency and high throughput can be achieved by choosing simple component codes and by exploiting the inherent parallelism of product codes.

Polynomial algebraic codes like Bose-Chaudhuri-Hocquenghem (BCH) codes [4] and Reed-Solomon (RS) codes [5] are able to detect t errors and identify their position in the codeword. They can be decoded efficiently with hard-decision algorithms; moreover, these algorithm can undergo substantial speed-up and complexity reduction when applied to particular codes. They have been considered in the past as component codes for product codes.

Forward error correction (FEC) schemes can incur an error-correction performance degradation at low BER. This sudden decrement of the BER curve slope is known as an error floor, which is usually caused by particular error patterns that are difficult for the decoder to correct. For a hard-decision product-code decoder, these patterns are often called *stall* patterns [6] or stopping sets.

To avoid the occurrence of an error floor, and thus meet extremely low BER requirements, post processing can be employed [7]. In this paper, we introduce a novel post-processing technique that dramatically increases the error-correction performance of polynomial product codes. Simulations show that the introduction of post-processing is able to lower the BER by three orders of magnitude.

The remainder of the paper is organized as follows. Section II introduces product-code encoding and decoding. Section III analyzes stall patterns and their contribution to the error floor. The proposed post-processing technique is detailed

in Section IV, and its impact on the error-correction performance is evaluated in Section V. Conclusions are drawn in Section VI.

II. PRODUCT CODES

Product codes are a class of error-correcting codes constructed through parallel concatenation. They were introduced in [1] and generalized in [8]. The encoding process starts by forming a matrix of information symbols; then, the rows of the matrix are encoded using the row component code. Afterwards, the columns of the matrix are encoded using the column component code. FEC schemes based on code concatenation [2], [9]–[12] have been widely adopted in the past in applications targeting very low BER. These schemes intertwine simpler codes greatly enhancing their error-correction performance. Reliable and well studied algebraic codes like BCH and RS codes have been used as component codes for product codes for decades.

Let us define a generic polynomial algebraic code (n, k, t) , where n is the code length in symbols, k the number of information symbols, and t the number of errors that the code can detect and correct. Given a generic matrix of $k_2 \times k_1$ symbols, a product code matrix can be obtained by encoding the rows with (n_1, k_1, t_1) , and the columns with (n_2, k_2, t_2) . For simplicity, in the remainder of the paper we assume that both rows and columns have been encoded with the same component code (n, k, t) : the generalization of the presented results is straightforward.

Product-code iterative hard-decision decoding was first described in [13]. The decoding process follows the same schedule as product-code encoding: first, the rows are decoded with the row component decoder, and then the columns are decoded with the column component decoder. The process is repeated for a set number of iterations. The decoding latency depends on the number of iterations and on the complexity of the component decoding. Moreover, product codes allow for a high degree of parallelism in the decoder, since all rows (columns) of the product-code matrix can be decoded concurrently: consequently, the hardware architecture of the decoder plays an important role in determining the decoding latency.

On the other hand, the error-correction performance of a product code mainly depends on the choice of the component codes, together with their length and rate. Increasing the number of decoding iterations can substantially improve the error-

correction performance at high to medium BER. However, this technique will prove mostly ineffective at very low BER.

III. ERROR FLOOR AND STALL PATTERN ANALYSIS

Error-correcting codes can incur a flattening of the BER curve at low BER, that cannot be overcome by improving the channel conditions or increasing the number of decoding iterations. This degradation of the error-correction performance is called an error floor. An error floor is usually caused by combinations of errors that are hard to detect and correct. In the specific case of product codes decoded with hard-decision algorithms, these are known as stall patterns [6]. Since polynomial codes are able to detect and correct t errors, stall patterns are defined as follows. Let us consider a generic product code matrix. Let r_i be the set of symbols in the row component codeword i , and c_j be the symbols in the column component codeword j .

Definition 1: A stall pattern is a set S of codeword symbols with the following properties:

- 1) If $s_{i,j} \in S$ then $|r_i \cap S| > t, \forall(i, j)$,
- 2) If $s_{i,j} \in S$ then $|c_j \cap S| > t, \forall(i, j)$,

where $s_{i,j}$ is the symbol at the intersection of row i and column j .

To perform our analysis, we pessimistically assume that decoding fails if a stall pattern exists in the received frame, that is we neglect the possibility that undetected errors in the component decoders cause the product decoder to avoid the stall [6]. To obtain a lower bound, we can consider the probability that a minimal stall pattern occurs in the received frame, i.e. error patterns S for which $s_{i,j} \in S \Rightarrow |r_i \cap S| = t + 1$ and $|c_j \cap S| = t + 1$. We expect the error floor to be dominated by stall patterns present in the received frame [6], [11], and the probability of occurrence of a stall pattern to be inversely proportional to its cardinality. Therefore such a lower bound should be reasonably tight.

To count minimal stall patterns, we first select the $t + 1$ rows that will be affected, and then select the $t + 1$ column positions that will be shared by all the affected rows. The number M of minimal patterns is thus given by

$$M = \binom{n}{t+1}^2. \quad (1)$$

We consider bit errors to be independent: thus, for a binary code, the probability that a minimal stall pattern occurs in the received vector at specific bit positions is $p^{(t+1)^2}$, where p is the channel BER. The probability of observing any minimal stall pattern in the received vector is then $Mp^{(t+1)^2}$. When a decoding failure due to a minimal stall occurs, $(t + 1)^2$ of the n^2 bits in the frame are in error, and therefore the BER is lower bounded by

$$f_{\min}(p) = \frac{(t + 1)^2 Mp^{(t+1)^2}}{n^2}. \quad (2)$$

If instead the component codes are non-binary codes able to correct up to t symbols of b bits, we can obtain a lower bound on the Symbol Error Rate (SER) by taking $f_{\min}(p_s)$, where

p_s is the channel SER and is given by $p_s = 1 - (1 - p)^b$ for binary memoryless channels. For the non-binary case, the bound $g_{\min}(p)$ on the BER can be expressed as

$$g_{\min}(p) = \frac{(b - 1)p + 1}{b^2} f_{\min}(1 - (1 - p)^b), \quad (3)$$

where $(b - 1)p + 1$ is the expected number of wrong bits in an erroneous symbol.

Stall-pattern avoidance can greatly benefit from the use of *extended*-polynomial codes. An extended binary polynomial code of length $n + 1$ is composed of a binary polynomial code of length n and of an additional parity bit. While this additional parity bit does not improve the number of errors t that can be corrected by the code, it increases to $t + 1$ the errors that can be detected, with a small cost in terms of code rate. Consequently, a product code based on extended-polynomial codes can detect decoding failures caused by minimal stall patterns. Non-binary polynomial codes like RS can be extended by adding a b -bit parity symbol instead of a single parity bit. The parity symbol is the sum of all the codeword symbols performed over the Galois Field of order b . This means that the i^{th} bit of the parity symbol can be computed by calculating the parity of the i^{th} bit of every codeword symbol. As long as the component polynomial code is linear, every row and every column of the product code based on its extended version is a valid component codeword.

Algorithm 1 portrays how the additional parity bit can be used in the decoding of binary polynomial component codes. Without loss of generality, we assume that this parity bit is placed at position $n + 1$ in the codeword, and we identify it as r_{n+1} . The CD function refers to the standard component decoder, which returns a flag `FAIL` indicating whether or not the decoder detected a failure, and in case it succeeded a vector e of length n indicating the location of errors. The notation $x_{i:j}$ with $i \leq j$ refers to a vector of length $j - i + 1$ containing elements $i, i + 1, \dots, j$ of the vector x . The operator \oplus denotes modulo-2 addition. The extended-code decoder declares failure either in case `FAIL` is risen or in case $d = t$ and $pr \neq 0$, i.e. when t errors have been detected but the parity check has failed.

Algorithm 2 describes the decoding process in case of extended non-binary polynomial codes, where error vector e^{sym} identifies the wrong symbols.

IV. STALL PATTERN POST PROCESSING

The post-processing technique mentioned in [7] is applied to the binary erasure channel: all symbols of the row component codes whose decoding has failed are changed to erasures, then column component codes are decoded. Failed columns are changed to erasures as well, and the process is repeated until no failures are detected. We take a different approach to post processing. The structure of a product code guarantees that after a full decoding iteration (rows + columns), any bit in error is located at the intersection of a row codeword and of a column codeword that are both in error. Some error locations can thus be identified from the extended-code row and column

Algorithm 1: Decoding of extended binary polynomial codes.

input : Component codeword r
output: Updated codeword r'
begin
 FAIL, $e \leftarrow \text{CD}(r_{1:n})$
 if FAIL **then**
 $r' \leftarrow r$
 else
 $d := \sum_{i=1}^n e_i$
 $pr := (d + \sum_{i=1}^{n+1} r_i) \bmod 2$
 if $d < t$ **then**
 $r'_{1:n} \leftarrow r_{1:n} \oplus e$
 $r'_{n+1} \leftarrow r_{n+1} \oplus p$
 else if $pr = 0$ **then**
 $r'_{1:n} \leftarrow r_{1:n} \oplus e$
 $r'_{n+1} \leftarrow r_{n+1}$
 else
 $r' \leftarrow r$

Algorithm 2: Decoding of extended non-binary polynomial codes.

input : Component codeword r
output: Updated codeword r'
begin
 FAIL, $e^{\text{sym}}, e \leftarrow \text{CD}(r_{1:n})$
 if FAIL **then**
 $r' \leftarrow r$
 else
 $d := \sum_{i=1}^n e_i^{\text{sym}}$
 for $w = 1 : b$ **do**
 $d_w^b := \sum_{i=0}^{n-1} e_{ib+w}$
 $pr_w := (d_w^b + \sum_{i=0}^n r_{ib+w}) \bmod 2$
 if $d < t$ **then**
 $r'_{1:nb} \leftarrow r_{1:nb} \oplus e$
 $r'_{nb+1:nb+b} \leftarrow r_{nb+1:nb+b} \oplus pr_{1:b}$
 else if $pr_{1:b} = 0$ **then**
 $r'_{1:nb} \leftarrow r_{1:nb} \oplus e$
 $r'_{nb+1:nb+b} \leftarrow r_{nb+1:nb+b}$
 else
 $r' \leftarrow r$

decoding failures. Note however that bit positions located at the intersection of a row and of a column that are both in error are not necessarily in error. As shown below, newly introduced errors can be corrected by an additional decoding iteration.

We propose a post-processing algorithm that inverts the bits located at the intersection of rows and columns that are known to be in error. It is performed after a set number of product-code decoding iterations have been completed. Let us denote by R the set of row indices for which the extended-component decoder reported a decoding failure. Similarly we denote by C the set of column indices for which the extended-

Table I
STALL PATTERN MULTIPLICITY

n_e	$m(n_e)$
12	8
14	72
15	16
16	1

code decoding has failed.

- *Binary component codes*: if $|R| > 0$ and $|C| > 0$, we flip the bit located at the intersection of each row in R with each column in C . Since this may introduce new bit errors, we then perform one full product decoding iteration.
- *Non-binary component codes*: in this case, the intersection of a row and a column is a b -bit symbol. However, it is possible to identify which bits of the symbol are most probably wrong by using the parity symbols. Let us take row $i \in R$ and column $j \in C$, and recompute their respective parity symbols. A bit w in symbol (i, j) will be flipped if bit w of the row or column parity symbol is different from the corresponding bit in the recomputed parity symbol. For example, if the row i parity symbol has mismatching bits 1 and 3, and column j parity symbol has mismatching bits 3 and 4, we flip bits 1, 3 and 4 in symbol (i, j) . If instead neither row i nor column j have mismatching bits in their parity symbols, we flip all bits of symbol (i, j) . Like with binary component codes, a full product decoding iteration follows the bit flipping.

Since the codeword positions involved in the post processing are not guaranteed to be in error, it is non-trivial to determine which error patterns can indeed be resolved using this algorithm. It can correct minimal stall patterns, since all bits in error correspond to error-row and error-column intersections. On the other hand, stall patterns S such that $s_{i,j} \in S \Rightarrow |r_i \cap S| = t + 2$ and $|c_j \cap S| = t + 2$ are clearly not correctable, since none of the incorrect rows or columns can be detected.

For small values of t , it is possible to perform an exhaustive pattern search to determine which ones are correctable using post processing and which ones are not. Let $n_e = |S|$ be the cardinality of a stall pattern S , and let $m(n_e)$ be the number of patterns with cardinality n_e such that $s_{i,j} \in S \Rightarrow t < |r_i \cap S| \leq t + 2$ and $t < |c_j \cap S| \leq t + 2$. The BER of a binary code is then lower bounded by

$$f_{\text{pp}}(p) = \frac{M}{(n+1)^2} \sum_{n_e} m(n_e) \cdot p^{n_e} \cdot (1-p)^{(t+2)^2 - n_e} \cdot n_e, \quad (4)$$

where

$$M = \binom{n+1}{t+2}^2. \quad (5)$$

As for the analysis presented in Section III, in the case of non-binary codes, (4) becomes a lower bound on the SER if the channel BER p is replaced with the channel SER $p_s = 1 - (1-p)^b$. The BER lower bound for non-binary codes is

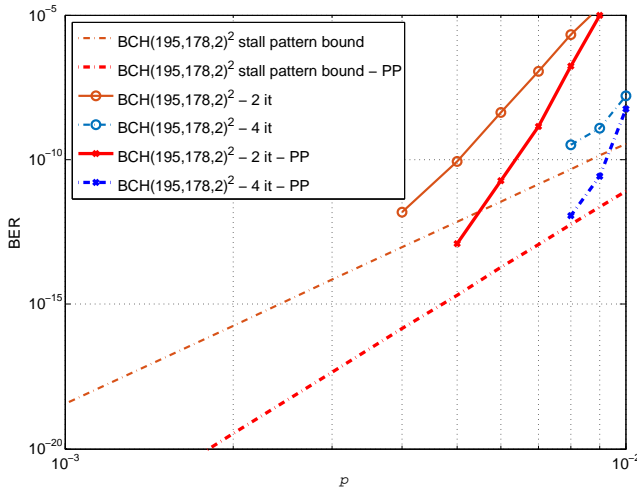


Figure 1. Stall-pattern contribution bounds and BER curves for extended BCH-based $(195,178,2)^2$ product code.

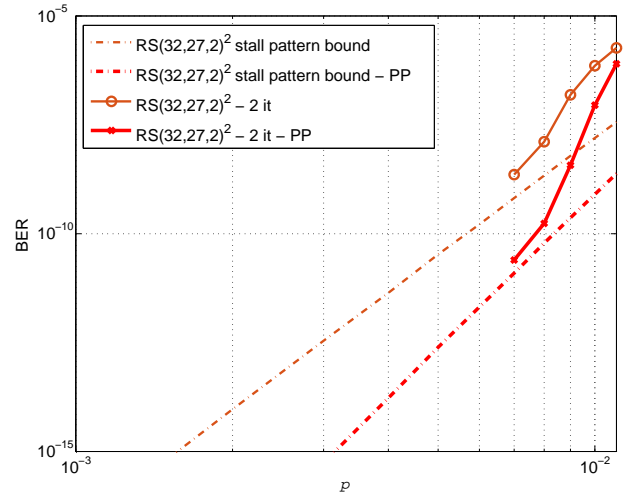


Figure 2. Stall-pattern contribution bounds and BER curves for extended RS-based $(32,27,2)^2$ product code.

instead expressed by

$$g_{pp}(p) = \frac{(b-1)p+1}{b^2} f_{pp}(1-(1-p)^b). \quad (6)$$

For $t = 2$, the non-zero values of $m(n_e)$ obtained using an exhaustive search are listed in Table I.

V. ERROR-CORRECTION PERFORMANCE

Figure 1 shows the BER and the stall-pattern contribution to the error floor for the product code based on the $(195, 178, 2)$ extended BCH code, with and without post processing, for two and four decoding iterations. It can be seen that the error-correction performance of the code is greatly enhanced for all values of p , and that the error bound is lowered and increased in steepness by post processing. At $p = 3 \times 10^{-3}$, the contribution of stall patterns to the error floor is decreased by more than three orders of magnitude. At the same time, post processing allows substantial BER gain under all decoding conditions. The tightness of the bound can be noticed in case of four decoding iterations and post processing, where the error floor is reached at higher p values.

Similar results are shown in Figure 2 for the $(32, 27, 2)$ extended RS-based product code, with $b = 5$. Two decoding iterations, with and without post processing, are sufficient to show how the BER curves closely follow the error bound. Post processing improves the error correction performance and decreases the stall-pattern contribution to the error floor by about two orders of magnitude.

VI. CONCLUSIONS

In this work we have proposed a novel post-processing technique for product codes based on extended-polynomial component codes. Post-processing uses the knowledge of failed row- and column-component-code decoding to flip the bits at their intersection. This technique was shown to be able to both greatly lower the product-code error bound and to increase its steepness: on an example code, the error floor is

lowered by more than three orders of magnitude at a channel BER of 2×10^{-3} . Simulation results show a comparable error correction performance improvement under various decoding conditions.

REFERENCES

- [1] P. Elias, "Error-free coding," *Trans. IRE Prof. Group Inf. Theory*, vol. 4, no. 4, pp. 29–37, September 1954.
- [2] Y.-Y. Jian, H. Pfister, K. Narayanan, R. Rao, and R. Mazahreh, "Iterative hard-decision decoding of braided BCH codes for high-speed optical communication," in *IEEE Global Commun. Conf. (GLOBECOM)*, Dec 2013, pp. 2376–2381.
- [3] R. Le Bidan, C. Leroux, C. Jégo, P. Adde, and R. Pyndiah, "Reed-Solomon turbo product codes for optical communications: From code optimization to decoder design," *EURASIP J. Wirel. Commun. Netw.*, vol. 2008, pp. 14:1–14:14, Jan 2008.
- [4] R. Bose and D. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Inf. Control*, vol. 3, no. 1, pp. 68 – 79, 1960.
- [5] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.
- [6] B. Smith, A. Farhood, A. Hunt, F. Kschischang, and J. Lodge, "Staircase codes: FEC for 100 Gb/s OTN," *J. Lightw. Technol.*, vol. 30, no. 1, pp. 110–117, Jan 2012.
- [7] S. Emmadi, K. R. Narayanan, and H. D. Pfister, "Half-product codes in flash memory," in *Non-Volatile Memories Workshop*, Mar 2015. [Online]. Available: <http://commtheory.wdfiles.com/local--files/publications/NVM2015.pdf>
- [8] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, Sep 1981.
- [9] A. Tychopoulos, O. Koufopavlou, and I. Tomkos, "FEC in optical communications - a tutorial overview on the evolution of architectures and the future prospects of outband and inband FEC for optical communications," *IEEE Circuits Devices Mag.*, vol. 22, no. 6, pp. 79–86, Nov 2006.
- [10] K. Lee and H. Lee, "A high-performance concatenated BCH code and its hardware architecture for 100 Gb/s long-haul optical communications," in *Int. SoC Design Conf. (ISOCC)*, Nov 2010, pp. 428–431.
- [11] J. Justesen, "Performance of product codes and related structures with iterated decoding," *IEEE Trans. Commun.*, vol. 59, no. 2, pp. 407–415, February 2011.
- [12] W. Chen and T. Dong, "Low complexity product codes with LDPC codes achieving ultra low BER," in *IEEE Int. Conf. Commun. Technol. (ICCT)*, Nov 2012, pp. 1312–1316.
- [13] N. Abramson, "Cascade decoding of cyclic product codes," *IEEE Trans. Commun. Technol.*, vol. 16, no. 3, pp. 398–402, June 1968.