Received: 21 November 2022 | Revised: 21 March 2023 | Accepted: 24 April 2023

DOI: 10.1002/eng2.12675

SHORT COMMUNICATION

Engineering Reports Open Access WILEY

Logarithmic time encoding and decoding of integer error control codes

Aleksandar Radonjic | **Vladimir Vujicic**

Institute of Technical Sciences of the Serbian Academy of Sciences and Arts, Belgrade, Serbia

Correspondence
Aleksandar Radonjic, Institute of Technical Sciences of the Serbian Academy of Sciences and Arts, Belgrade, Serbia.
Email: sasa_radonjic@yahoo.com

Funding information
The Ministry of Science, Technological Development and Innovation of the Republic of Serbia, Grant/Award Number: 451-03-47/2023-01/200175

Abstract

One of the most important characteristics of all error control codes (ECCs) is the complexity of the encoding/decoding algorithms. Today, there are many ECCs that can correct multiple bit errors, but at the price of high encoding/decoding complexity. Among the rare exceptions are integer ECCs (IECCs), whose serial encoding/decoding algorithms run in $O(n)$ time, where n is the codeword length. In this article, we show that IECCs can be encoded/decoded even faster, that is, that their parallel encoding/decoding algorithms have $O(log_2 n)$ time complexity.

KEYWORDS

decoding, encoding, integer error control codes, logarithmic time complexity

1 | INTRODUCTION

In order to find out whether one algorithm is more computationally efficient than the other, researchers use two models: one based on the random access machine (RAM) and the other based on parallel RAM.[1] The first model consists of a processor that has an unrestricted amount of memory and that can perform various operations on data bits. Unlike it, the parallel RAM is a model in which multiple processors perform operations in parallel and share a common unlimited amount of memory.

Using these models, the researchers were also investigating the time complexity of the encoding and decoding procedures for various error control codes (ECCs). The obtained results have shown that many codes, such as LDPC, Polar, Reed-Solomon (RS), and Turbo codes, are complex to encode/decode. In particular, in References 2–4 it was shown that LDPC codes can be encoded in linear or quasi-linear time, whereas their decoding algorithms run in linear or log-linear time.[5–7] Polar codes, on the other hand, can be encoded/decoded in log-linear time,[8,9] while the encoding/decoding procedures for RS codes have quasi-log-linear time complexity.[10] The most complicated of all ECCs are Turbo codes, since their encoding and decoding algorithms run in quasi-linear and quasi-exponential time, respectively.[11,12]

Although all the mentioned codes are complex to encode/decode, and therefore complicated to implement, they are used in various communication systems. So, for example, it is known that LDPC, Polar, and Turbo codes are applied in wireless communication systems, such as digital video broadcasting and cellular networks.[13] On the other hand, RS codes are standardized in a number of applications such as optical networks, satellite communications, and storage systems.[13] The reason for such a massive use of the mentioned ECCs lies in the fact that reliable data transmission is much more important than the price paid for it.

In this article, we will show that reliable communication can be achieved in a much simpler way if integer ECCs (IECCs) are used. These codes use integer arithmetic, which brings with it a number of advantages, such as the possibility of efficient implementation on general purpose processors (GPPs). In one of the previous papers, we showed that IECCs can be serially encoded/decoded in linear time.[14] In this article, we will show that IECCs can be encoded/decoded even faster, that is, that their parallel encoding/decoding algorithms run in logarithmic time. We believe this fact will make them very attractive for potential use in future communication and memory systems.

The organization of this article is as follows: Section 2 deals with the basic concepts of IECCs. The parallel encoding/decoding algorithms for this family of codes are described and evaluated in Sections 3 and 4, while Section 5 concludes the article.

2 | IECCS: CONSTRUCTION AND ERROR CONTROL

In Reference 15 it was pointed out that IECCs share many common features with checksum codes.[16] One of them is that the codeword consists of k data bytes and one check-byte (Figure 1). In the case of IECCs, the check-byte is computed as the sum of the products of the integer values of the data bytes and the coefficients C_i. However, the syndrome S of the received codeword is calculated as Reference 16, that is, as the difference in value between the newly calculated and the received check-byte. Both these facts are summarized in the following definitions.

Definition 1 (17). *Let $Z_{2^b-1}=\{0,1,\ldots,2^b-2\}$ be the ring of integers modulo 2^b-1 and let $B_i=\sum_{n=0}^{b-1}a_n\cdot 2^n$ be the integer representation of a b-bit byte, where $a_n\in\{0,1\}$ and $1\le i\le k$. Then, the code C (b, k, c), defined as*

$$C(b,k,c)=\left\{x\in Z_{2^b-1}^{k+1}\ :\ \sum_{i=1}^{k}C_i\cdot B_i\equiv B_{k+1}\pmod{2^b-1}\right\} \tag{1}$$

is an $(kb+b, kb)$ integer code, where $x=(B_1, B_2, \ldots, B_k, B_{k+1})\in Z_{2^b-1}^{k+1}$ is the codeword vector, $c=(C_1, C_2, \ldots, C_k, 1)\in Z_{2^b-1}^{k+1}$ is the coefficient vector and $B_{k+1}\in Z_{2^b-1}$ is an integer.

Definition 2 (17). *Let $x=(B_1, B_2, \ldots, B_k, B_{k+1})\in Z_{2^b-1}^{k+1}$, $y=(\underline{B}_1, \underline{B}_2, \ldots, \underline{B}_k, \underline{B}_{k+1})\in Z_{2^b-1}^{k+1}$ and $e=(\underline{B}_1-B_1, \underline{B}_2-B_2, \ldots, \underline{B}_k-B_k, B_{k+1}-\underline{B}_{k+1})=(e_1, e_2, \ldots, e_k, e_{k+1})\in Z_{2^b-1}^{k+1}$ be the transmitted codeword, the received codeword and the error vector, respectively. Then, the syndrome S of the received codeword is defined as*

$$S=\sum_{i=1}^{k}C_i\cdot\underline{B}_i-\underline{B}_{k+1}\pmod{2^b-1}=\sum_{i=1}^{k+1}e_i\cdot C_i\pmod{2^b-1}. \tag{2}$$

From (2) it is easy to see that the nonzero value of S indicates the presence of one or more errors within t b-bit bytes ($1\le t<k+1$). The decoder will be able to correct these errors if the corresponding IECC is constructed through the following steps.

1. *Defining the error type that the code should correct*. In essence, we need to define the values of t and e_i. For instance, if we want to construct a class of codes that can correct single errors within one b-bit byte, the values of t and e_i will be equal to $t=1$ and $e_i=\pm 2^r$, where $0\le r\le b-1$. On the other hand, if we want to construct a class of codes capable of correcting single errors within two b-bit bytes, the values of t and e_i will be equal to $t=2$ and $e_i=\pm 2^r$, where $0\le r\le b-1$ (Table 1).

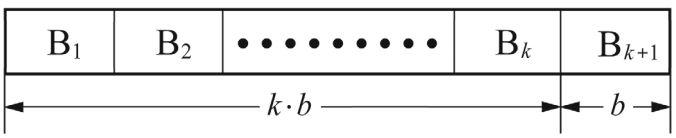

FIGURE 1 The codeword structure for IECCs

TABLE 1 The main characteristics of several classes of IECCs

IECCs	t	e_i	$k_{\max}$	$\lvert\xi\rvert$
Codes from Reference 15	1	$\{\pm 2^r : 0 \le r \le b-1\}$	$\left\lfloor \frac{2^{b-1}-b-1}{b} \right\rfloor$	$2 \cdot b \cdot (k+1)$
Codes from Reference 18	1	$\{\pm 2^r \pm 2^s : 0 \le r < s \le b-1\}$	$\left\lfloor \frac{2^{b-1}-(b-1)^2}{(b-1)^2-1} \right\rfloor$	$\left[2 \cdot (b-1)^2 - 2\right] \cdot (k+1)$
Codes from Reference 17	1 2	$\{\pm 2^r \pm 2^s : 0 \le r < s \le b-1\}$ $\{\pm 2^r : 0 \le r \le b-1\}$	$\left\lfloor \frac{2^{(b-1)/2}-b+1}{b} \right\rfloor$	$2 \cdot [b \cdot (k+1)-1]^2 - 2$
Codes from Reference 19	2	$\{\pm 2^r : 0 \le r \le b-1\}$	$\left\lfloor \frac{\sqrt{2^{b+1}+(b-1)^2-4}-b-1}{2 \cdot b} \right\rfloor$	$2 \cdot b \cdot (b \cdot k+1) \cdot (k+1)$

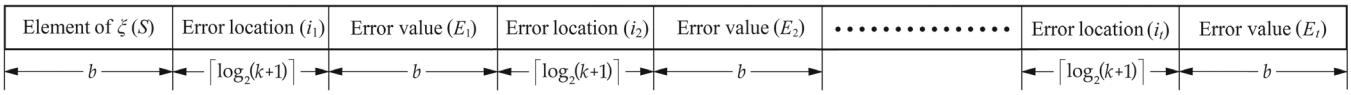

FIGURE 2 Bit-width of one ST entry for general IECCs

2. *Defining the set of correctable syndromes*. In the general case, this set is defined as

$$\xi = \bigcup_{h=1}^{t} s_h, \tag{3}$$

where

$$s_1 = \left\{ e_{i_1} \cdot C_{i_1} \pmod{2^b - 1} : 1 \le i_1 \le k+1 \right\}, \tag{4}$$

$$s_2 = \left\{ e_{i_1} \cdot C_{i_1} + e_{i_2} \cdot C_{i_2} \pmod{2^b - 1} : 1 \le i_1 < i_2 \le k+1 \right\}, \tag{5}$$

$$\vdots$$

$$s_t = \left\{ e_{i_1} \cdot C_{i_1} + e_{i_2} \cdot C_{i_2} + \cdots + e_{i_t} \cdot C_{i_t} \pmod{2^b - 1} : 1 \le i_1 < i_2 < \cdots < i_t \le k+1 \right\}. \tag{6}$$

3. *Finding the coefficients C_i*. For each value of $b \ge 2$ it is necessary to perform a computer search to find the coefficients C_i. Although the number of coefficients increases with increasing b, the upper theoretical limit ($k_{\max}$), in the general case, cannot be determined (the value of $k_{\max}$ depends on the class of IECCs) (Table 1). Regardless of that fact, the values of the coefficients C_i must be such that

$$s_1 \cap s_2 \cap \cdots \cap s_t = \emptyset,$$
$$|\xi| = \sum_{h=1}^{t} |s_h| \cdot \binom{k+1}{h},$$

where $|X|$ denotes the cardinality of X.

4. *Selecting the code parameters and generating the syndrome table*. The number of the coefficients found determines the number of b-bit bytes that can be protected. By choosing whether to use all coefficients or not, we determine the size of the codeword as well as the size of the syndrome table (ST). The ST always has $|\xi|$ entries and is generated based on the values of t, b, k, e_i, and C_i. The purpose of each entry is to describe the relationship between the nonzero syndrome, error locations and error values (Figure 2).

From the above steps, it is clear that the IECC construction process is independent of the encoding/decoding process. However, for the sake of completeness it is needed to point out that the communication between end-points starts only when the ST is generated and stored in local memories. In that case, for each incoming codeword, the decoder will calculate the syndrome S. If its value is equal to zero ($S=0$), the decoder will assume that the codeword is error-free. However, if the value of S is nonzero ($S\neq 0$), the decoder will lookup the ST in order to find the entry with the first b bits as that of the syndrome S. If such an entry exists, the decoder will perform (in parallel) the operations:

$$B_{i_1} = \underline{B}_{i_1} + E_1 \left(\mathrm{mod}\ 2^b - 1\right), \tag{7}$$

$$B_{i_2} = \underline{B}_{i_2} + E_2 \left(\mathrm{mod}\ 2^b - 1\right), \tag{8}$$

$$\vdots$$

$$B_{i_t} = \underline{B}_{i_t} + E_t \left(\mathrm{mod}\ 2^b - 1\right). \tag{9}$$

Otherwise, it will declare an uncorrectable error.

3 | PARALLEL ENCODING AND DECODING OF IECCS

In Reference 14 it was shown that the serial encoding/decoding algorithms for IECCs have linear time complexity. However, the data can also be processed in parallel. The motivation for such an approach lies in the concept of parallel addition of p integers. In particular, if a binary tree structure is used, the addition of p integers can be performed in $O(log_2 p)$ time[1] (Figure 3). Using this fact, we can state the following theorems.

Theorem 1. *Any (kb + b, kb) IECC can be encoded in parallel in $O(log_2 n)$ time.*

Proof. Let us analyze the expression (1). The first thing we notice is that the check-byte is computed as the sum of k products. Each of these products is calculated independently (Figure 4A), which means that the encoder must perform $b{\cdot}log_2\ b$ bit operations[20] in order to calculate the product $N_i = C_i{\cdot}B_i$, where $i=1, 2, \ldots, k$. After that, the encoding procedure reduces to modular addition of k integers using a binary tree with $\lceil log_2 k\rceil$ levels. This means that the check-byte B_{k+1} will be computed after $\lceil log_2 k\rceil$ additions, where each addition takes b bit operations. Given this and the fact that the codeword has $n=(k+1){\cdot}b$ bits, from the expression

$$O\left(b\cdot log_2 b + b\cdot \lceil log_2 k\rceil\right) \approx O\left(b\cdot log_2(b\cdot k)\right) \approx O\left(b\cdot log_2 n\right) = b\cdot O\left(log_2 n\right) = const.\cdot O\left(log_2 n\right) = O\left(log_2 n\right)$$

it is clear that any IECC can be encoded in parallel in logarithmic time. □

Theorem 2. *Any (kb + b,kb) IECC can be decoded in parallel in $O(log_2 n)$ time.*

Proof. The decoding process for all IECCs consists of three steps: calculating the syndrome S, looking up the ST and correcting the errors. From (2) we see that performing the first step requires only one operation more than the encoding process. However, if we parallelize all the calculations (Figure 4B), we easily come to the conclusion that the syndrome S will be computed after $b{\cdot}log_2 b + b{\cdot}\lceil log_2(k+1)\rceil$ binary operations. If the value of S is nonzero, the decoder will lookup the ST to get the error correction data. Since the ST can be presorted in ascending order (according to the values of S), it is possible to use a binary search algorithm.[1] In that case, the number of table lookups (TLs) will not be greater than $\lfloor log_2|\xi|\rfloor + 2$[14] where each TL takes b bit operations (the comparison of two b-bit integers). If we add to this the fact that the last step (error correction) requires b bit operations (t integer additions in parallel) and that the value of $|\xi|$ is never greater than 2^b-2, we get the inequality

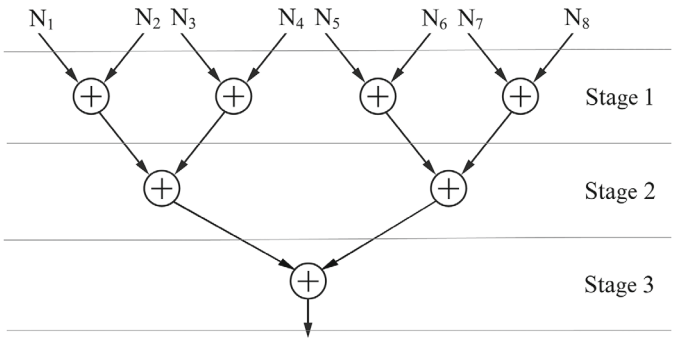

FIGURE 3 Illustration of the binary tree addition algorithm

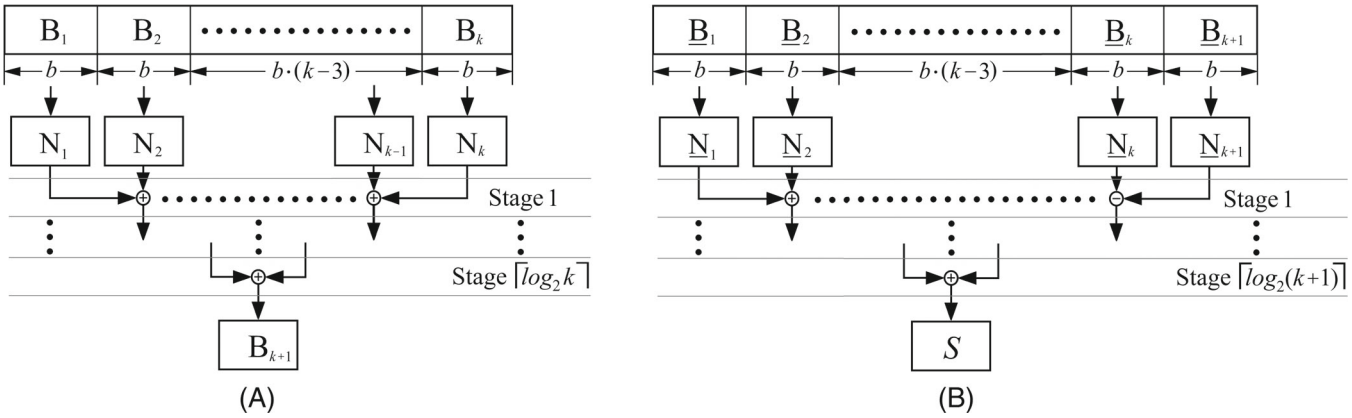

FIGURE 4 Illustration of the parallel algorithm for (A) encoding and (B) syndrome computing

$$O\left(b\cdot log_2 b + b\cdot\lceil log_2(k+1)\rceil + b\cdot\lfloor log_2|\xi|\rfloor + 3\cdot b\right) < O\left(b\cdot log_2[b\cdot(k+1)] + b\cdot\left\lfloor log_2\left(2^b-2\right)\right\rfloor + 3\cdot b\right) <$$
$$< O\left(b\cdot log_2 n + b^2 + 3\cdot b\right) = b\cdot O\left(log_2 n + b + 3\right) = const.\cdot O\left(log_2 n + const.\right) = O\left(log_2 n\right)$$

from which it is clear that any IECC can be decoded in parallel in logarithmic time. □

4 | EVALUATION

In the previous section, we have seen that the complexity of encoding/decoding of IECCs does not depend on the code's strength. This, however, is not the case with standard ECCs. An obvious example are LDPC codes, whose performance depends both on the code type and the decoding algorithm used. This is the reason why it is often stated that algorithms for decoding weaker LDPC codes run in $O(n)$ time,[5] while those used for decoding stronger LDPC codes have $O(n\cdot log_2 n)$ complexity.[6,7] On the other hand, it is known that all LDPC codes can be encoded in $O(n)$ time.[4] As for Polar codes, they can be encoded and decoded in $O(n\cdot log_2 n)$ and $O(L\cdot n\cdot log_2 n)$ time, respectively, whereby the decoder performance increases with the list size L.[8,9] Unlike LDPC and Polar codes, the encoding/decoding complexity of RS codes grows with the number of check bytes. In particular, if the number of check bytes r is even, RS codes can be encoded and decoded in $O(n\cdot log_2 r)$ and $O(n\cdot log_2 r + r\cdot log_2{}^2 r)$ time, respectively.[10] The fourth and most complex ECCs are Turbo codes. According to References 11,12, these codes can be encoded and decoded in $O(n\cdot m)$ and $O(n\cdot 2^m)$ time, respectively, where $m+1$ is the constraint length of the convolutional codes (Table 2).

In addition to having high encoding/decoding complexity, the mentioned codes are very slow when implemented in software. The reason for this lies in the fact that they use finite field (FF) arithmetic, which is entirely different from the integer and floating point (FP) arithmetic of GPPs. Since the emulation of FF operations requires a large number of instructions[21] (thus slowing down the performance of the processor),

TABLE 2 Comparison of various ECCs

Codes	Lowest encoding complexity	Lowest decoding complexity	Preferred type of implementation
All IECCs	$O(log_2 n)$	$O(log_2 n)$	Software
LDPC codes	$O(n)$	$O(n)$	Hardware
RS codes	$O(n \cdot log_2 r)$	$O(n \cdot log_2 r + r \cdot log_2{}^2 r)$	Hardware
Polar codes	$O(n \cdot log_2 n)$	$O(n \cdot log_2 n)$	Hardware
Turbo codes	$O(n \cdot m)$	$O(n \cdot 2^m)$	Hardware

TABLE 3 Highest decoding speeds for several software-based decoders

Codes	Type of processor	Number of cores	Code parameters	Decoding throughput
LDPC code[25]	GPP	20	(16384, 4096)	11.25 Gbps
RS code[23]	GPP + GPU	22 + 3072	(2040, 1784)	10.65 Gbps
Polar code[22]	GPP	4	(2048, 1707)	2.17 Gbps
Turbo code[24]	GPP	12	(18432, 6144)	1.7 Gbps

some researchers decided to use extremely powerful GPPs and/or graphical processing units (GPUs). However, even this very expensive approach has not proven to be applicable[22–25] in future communication networks (Table 3).

Unlike FF-based codes, IECCs are perfectly suited for implementation on 64-bit processors. This feature is not only related to the fact that GPPs have four integer units (IUs) per core, but also that each IU operates independently of the other ones (Figure 5).[26] This means that the proposed encoding/decoding algorithms can be fully implemented if the total number of IUs is not less than $k+1$. In that case, the encoder (GPP) would take $N_{\mathrm{IM}} + \lceil log_2 k \rceil \cdot N_{\mathrm{IA}}$ clock cycles to generate the check byte B_{k+1}, where N_{IM} and N_{IA} denote the number of clock cycles needed to perform one integer multiplication and one integer addition, respectively. Starting from the fact that the equalities $N_{\mathrm{IM}} = 3$ and $N_{\mathrm{IA}} = 1$ apply to all GPPs,[26] we easily come to the conclusion that the encoder can process

$$G_{EN} = \frac{\text{clock speed} \times \text{dataword length}}{\text{number of clock cycles}} = \frac{\text{clock speed} \cdot k \cdot b}{\lceil log_2 k \rceil + 3} \text{ bits per second.} \tag{10}$$

In a similar way it can be shown that the decoder processes

$$G_{DE} = \frac{\text{clock speed} \times \text{codeword length}}{\text{number of clock cycles}} = \frac{\text{clock speed} \cdot (k+1) \cdot b}{\lceil log_2 (k+1) \rceil + \left(\lfloor log_2 |\xi| \rfloor + 2 \right) \cdot N_{\mathrm{ST}} + 5} \text{ bits per second,} \tag{11}$$

where N_{ST} denotes the number of clock cycles that the decoder needs to access the ST (this table must be stored in the local GPP's memory).

If we analyze the above expressions, we will notice that the encoding speed increases with increasing clock speed and/or codeword length. On the other hand, the decoding speed depends on four parameters, of which N_{ST} plays a dominant role (Table 4). This fact points to the conclusion that the ST should always be stored in the L1/L2 cache. If this is not feasible at the start, the size of the ST should be reduced by shortening the codeword length.

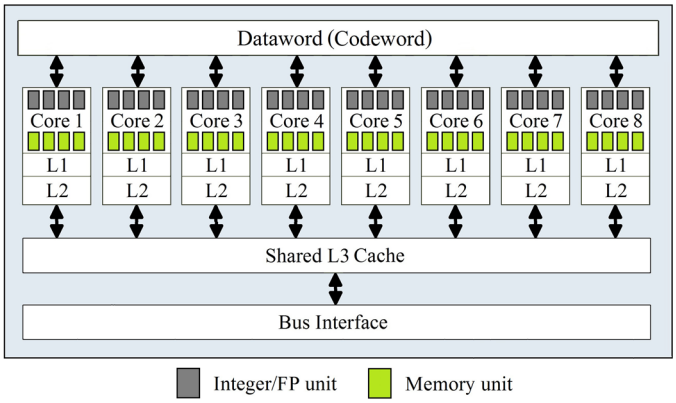

FIGURE 5 Block diagram of an eight-core GPP processing a dataword (codeword)

TABLE 4 Theoretical encoding/decoding throughputs for some 64-bit IECCs implemented on eight-core GPPs

Code parameters	k	$\lvert\xi\rvert$	Clock speed	Theoretical encoding throughput	Theoretical decoding throughput		
					$N_{ST} = 4$[a] (L1 cache)	$N_{ST} = 12$[a] (L2 cache)	$N_{ST} = 25$[a] (L3 cache)
(1920, 1856)	29	2^{12}	$3.0 \cdot 10^9$ Hz	696.0 Gbps	87.3 Gbps	32.4 Gbps	16.0 Gbps
(1920, 1856)	29	2^{13}	$3.0 \cdot 10^9$ Hz	696.0 Gbps	82.3 Gbps	30.3 Gbps	15.0 Gbps
(1920, 1856)	29	2^{14}	$3.0 \cdot 10^9$ Hz	696.0 Gbps	77.8 Gbps	28.5 Gbps	14.0 Gbps
(1920, 1856)	29	2^{12}	$3.5 \cdot 10^9$ Hz	812.0 Gbps	101.8 Gbps	37.8 Gbps	18.7 Gbps
(1920, 1856)	29	2^{13}	$3.5 \cdot 10^9$ Hz	812.0 Gbps	96.0 Gbps	35.4 Gbps	17.5 Gbps
(1920, 1856)	29	2^{14}	$3.5 \cdot 10^9$ Hz	812.0 Gbps	90.8 Gbps	33.3 Gbps	16.4 Gbps
(1984, 1920)	30	2^{12}	$3.0 \cdot 10^9$ Hz	720.0 Gbps	90.1 Gbps	33.4 Gbps	16.5 Gbps
(1984, 1920)	30	2^{13}	$3.0 \cdot 10^9$ Hz	720.0 Gbps	85.0 Gbps	31.3 Gbps	15.5 Gbps
(1984, 1920)	30	2^{14}	$3.0 \cdot 10^9$ Hz	720.0 Gbps	80.4 Gbps	29.5 Gbps	14.5 Gbps
(1984, 1920)	30	2^{12}	$3.5 \cdot 10^9$ Hz	840.0 Gbps	105.2 Gbps	39.0 Gbps	19.3 Gbps
(1984, 1920)	30	2^{13}	$3.5 \cdot 10^9$ Hz	840.0 Gbps	99.2 Gbps	36.5 Gbps	18.0 Gbps
(1984, 1920)	30	2^{14}	$3.5 \cdot 10^9$ Hz	840.0 Gbps	93.8 Gbps	34.4 Gbps	16.9 Gbps
(2048, 1984)	31	2^{12}	$3.0 \cdot 10^9$ Hz	744.0 Gbps	93.1 Gbps	34.5 Gbps	17.1 Gbps
(2048, 1984)	31	2^{13}	$3.0 \cdot 10^9$ Hz	744.0 Gbps	87.8 Gbps	32.3 Gbps	16.0 Gbps
(2048, 1984)	31	2^{14}	$3.0 \cdot 10^9$ Hz	744.0 Gbps	83.0 Gbps	30.4 Gbps	15.0 Gbps
(2048, 1984)	31	2^{12}	$3.5 \cdot 10^9$ Hz	868.0 Gbps	108.6 Gbps	40.3 Gbps	19.9 Gbps
(2048, 1984)	31	2^{13}	$3.5 \cdot 10^9$ Hz	868.0 Gbps	102.4 Gbps	37.7 Gbps	18.6 Gbps
(2048, 1984)	31	2^{14}	$3.5 \cdot 10^9$ Hz	868.0 Gbps	96.9 Gbps	35.5 Gbps	17.5 Gbps

[a] Typical number of clock cycles that a processor needs to access the L1/L2/L3 cache.[26]

5 | CONCLUSION

In this article, we have proposed algorithms for parallel encoding/decoding of IECCs. We have shown that the proposed algorithms have logarithmic time complexity and are perfectly suited for implementation on MPs. Both of these features can be used not only to improve the performance of existing codes, but also to construct new ones that would have the potential to be used in future communication and memory systems.

AUTHOR CONTRIBUTIONS

Aleksandar Radonjic: Writing - original draft preparation; writing - review and editing; conceptualization (equal); investigation (equal); validation (equal). **Vladimir Vujicic:** Conceptualization (equal); investigation (equal); validation (equal).

FUNDING INFORMATION

This article was supported by the Ministry of Science, Technological Development and Innovation of the Republic of Serbia (Grant No. 451-03-47/2023-01/200175).

CONFLICT OF INTEREST STATEMENT

The authors have no conflict of interest relevant to this article.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no datasets were generated or analysed during the current study.

ORCID

Aleksandar Radonjic https://orcid.org/0000-0003-3715-468X

REFERENCES

1. Miller R, Boxer L. *Algorithms Sequential & Parallel: A Unified Approach*. Cengage Learning; 2013.
2. Richardson T, Urbanke R. Efficient encoding of low-density parity check codes. *IEEE Trans Inf Theory*. 2001;47(2):638-656.
3. Lu J, Moura J. Linear Time Encoding of LDPC Codes. *IEEE Trans Inf Theory*. 2010;56(1):233-249.
4. Nozaki T. Parallel encoding algorithm for LDPC codes based on block-diagonalization. Proceedings of the IEEE International Symposium on Information Theory (ISIT'15); 2015:1911-1915.
5. Burshtein D. Iterative Approximate Linear programming decoding of LDPC codes with linear complexity. *IEEE Trans Inf Theory*. 2009;55(11):4835-4859.
6. Frolov A, Zyablov V. On the multiple threshold decoding of ldpc codes over GF(q). *Adv Math Commun*. 2017;11(1):123-137.
7. Rybin P, Andreev K, Zyablov V. Error exponents of LDPC codes under low-complexity decoding. *Entropy*. 2021;23(2):253.
8. Arıkan E. Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Trans Inf Theory*. 2009;55(7):3051-3073.
9. Li B, Shen H, Tse D. An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check. *IEEE Commun Lett*. 2012;16(12):2044-2047.
10. Tang N, Lin Y. Fast encoding and decoding algorithms for arbitrary (n, k) Reed-Solomon codes over F_2^m. *IEEE Commun Lett*. 2020;24(4):716-719.
11. Pei R, Wang Z, Huang Q, Wang J. Low complexity SOVA for turbo codes. *China Commun*. 2017;14(8):33-40.
12. Mohammed M, Abdessadek A. Performance and complexity comparisons of Polar codes and Turbo codes. Proceedings of the International Conference on Advanced Intelligent Systems for Sustainable Development (AI2SD'18); 2019:434-443.
13. Benvenuto N, Cherubini G, Tomasin S. *Algorithms for Communications Systems and Their Applications, 2nd Edition*. John Wiley and Sons Ltd.; 2021.
14. Radonjic A, Vujicic V. Integer codes correcting burst errors within a byte. *IEEE Trans Comput*. 2013;62(2):411-415.
15. Radonjic A. (Perfect) Integer codes correcting single errors. *IEEE Commun Lett*. 2018;22(1):17-20.
16. Maxino T, Koopman P. The effectiveness of checksums for embedded control networks. *IEEE Trans Depend Secure Comput*. 2009;6(1):59-72.
17. Radonjic A. Integer codes correcting double errors and triple-adjacent errors within a byte. *IEEE Trans Very Large Scale Integr (VLSI) Syst*. 2020;28(8):1901-1908.
18. Radonjic A, Vujicic V. Integer codes correcting sparse byte errors. *Cryptogr Commun*. 2019;11(5):1069-1077.
19. Radonjic A. Integer codes correcting single errors within two bytes. *J Circuits Syst Comput*. 2021;30(14):2150260.
20. Harvey D, Hoeven J. Integer multiplication in time $O(n\log n)$. *Ann Math*. 2021;193(2):563-617.
21. Wu Z, Gong C, Liu D. Computational complexity analysis of FEC decoding on SDR platforms. *J Signal Process Syst*. 2017;89(2):209-224.
22. Le Gal B, Leroux C, Jego C. Multi-Gb/s software decoding of polar codes. *IEEE Trans Signal Process*. 2015;63(2):349-359.
23. Suzuki T, Kim SY, Kani JI, Hanawa T, Suzuki KI, Otaka A. Demonstration of 10-Gbps real-time Reed–Solomon decoding using GPU direct transfer and kernel scheduling for flexible access systems. *J Lightw Technol*. 2018;36(10):1875-1881.
24. Le Gal B, Jego C. Low-latency and high-throughput software turbo decoders on multi-core architectures. *Ann Telecommun*. 2020;75(1–2):27-42.

25. Pignoly V, Le Gal B, Jego C, Gadat B, Barthe L. Fair comparison of hardware and software LDPC decoder implementations for SDR space links. Proceedings of the 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS); 2020:1-4; IEEE.
26. Fog A. *The Microarchitecture of Intel, AMD and via CPUs: An Optimization Guide for Assembly Programmers and Compiler Makers*. Technical University of Denmark; 2022. https://www.agner.org/optimize/microarchitecture.pdf

How to cite this article: Radonjic A, Vujicic V. Logarithmic time encoding and decoding of integer error control codes. *Engineering Reports*. 2023;e12675. doi: 10.1002/eng2.12675