Forward Error Correction with RaptorQ Code on Embedded Systems

T. Mladenov and K. Kim
Dept. Information & Communications,
Gwangju Institute of Science and Technology,
Gwangju 500-712, South Korea,
Email: {todor,kskim}@gist.ac.kr,

S. Nooshabadi
Department of Electrical and Computer Engineering,
Michigan Technological University,
Houghton, MI,
Email: saeid@mtu.edu

***Abstract*—Raptor code, a member of the rateless fountain codes family, has been a preferred technology for the forward error correction (FEC) at the application layer. It is used by the 3^{rd} Generation Partnership Program (3GPP) Multimedia Broadcast/Multicast Service (MBMS) and Digital Video Broadcasting (DVB-H) standards for multimedia broadcast and content delivery. Currently, a next generation fountain code, called RaptorQ, is being proposed. It aims at reducing to a minimum the redundant FEC information, outperforming significantly Raptor code. The improved coding performance comes at the expense of increased encoding and decoding complexity. This paper compares the coding properties, the decoding performance and the energy efficiency of the two codes on embedded system. Furthermore, simulations are performed for a practical MBMS scenario. Finally, conclusions are drawn with respect to the applicability of the new code for realtime multimedia broadcasting and content delivery.**

I. Introduction

The demand for mobile multimedia services is rapidly increasing due to the increase in the mobile Internet traffic. The growing number of clients, requiring the access to the same content at the same time, led to the specification of a point-to-multipoint Multimedia Broadcast/Multicast Service (MBMS), by the 3^{rd} Generation Partnership Program (3GPP) group [1]. MBMS is designed as an upgrade to the existing point-to-point packet based infrastructure of second and third generation mobile wireless networks. The forward error correction (FEC) schemes at the physical layer cannot provide sufficient reliability, due to heterogeneous system and channel conditions. In order to increase the quality and reduce the negotiation traffic between the sender and receiver, MBMS provides a second, application layer FEC.

Raptor code [2] is the preferred coding technique used for that purpose, mainly due to its rateless property, the ability to generate as many repair symbols as needed on the fly, the linear encoding and decoding complexity and performance independent of the channel erasure rate. Additionally, Raptor code is systematic. Recently, the next generation of Raptor code, called RaptorQ code [3], has been introduced. It offers better coding performance in terms of reduced symbol overhead for guaranteeing successful decoding and it supports larger source symbol block sizes.

Raptor code for application layer FEC is defined in [4]. The performance, energy profile and resource implication of Raptor decoding on a system on a chip (SoC) platform with an embedded processor are investigated in [5]. The trade-offs for the implementation of systems employing MBMS on embedded platform with Raptor code FEC are studied in [6].

This paper focuses on evaluating the performance of the software implementation of RaptorQ code on embedded system with the inactivation decoding Gaussian elimination (IDGE) algorithm specified in [3]. The results are compared to those of Raptor decoding with the IDGE algorithm in [4], for various source block sizes and symbol sizes. Additionally, the energy profiles of the two codes are presented and analyzed.

The chosen embedded system platform is a NIOS II soft-core processor, running on an Altera Stratix FPGA with the microprocessor clocked at 100 MHz and a DDR2 memory working at 400 MHz. Fig. 1 presents the high level block diagram of the embedded system on a chip (SoC) platform for the implementation of Raptor codes.

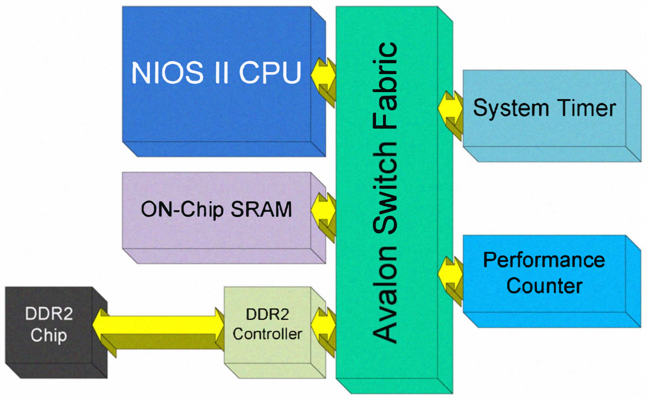

Fig. 1. RaptorQ codes on NIOS II embedded system.

The paper is organized as follows. Section II briefly outlines the structure of RaptorQ code. The IDGE algorithm is described in Section III. The complexity comparison between RaptorQ and Raptor codes and their energy efficiency are presented in Section IV. Section V concludes the paper.

II. RaptorQ Code

RaptorQ code [3] is a linear block code and as such it can be represented by its generator matrices. A block diagram of *Systematic RaptorQ Encoder/Decoder* is shown in Fig. 2.

Let $\mathbf{t}$ be a vector of K source symbols that are to be encoded ($1 \leq K \leq 56,403$). The size of each source symbol T varies from 1 to 1024 bytes for the usual practical cases. An arbitrary vector $\mathbf{t}$ of size K is padded with zeros to a vector $\mathbf{t}'$ of size K'. This operation is performed by the "Padding" block in Fig. 2. The size of K' can be any value from a subset of 477 source block size values, distributed in the range from 1 to $56,403$, for which RaptorQ code is defined. The mapping of K to K' minimizes the amount of table information that needs to be stored and enables faster encoding and decoding. Additionally, the padding zero symbols are not transmitted but they are a-priori known at the decoder and act as an additionally available parity information. Vector $\mathbf{d}$, at the input of the pre-code, consists of L symbols: the K' padded source symbols in vector $\mathbf{t}'$ plus $S + H$ zero symbols in vector $\mathbf{z}$.

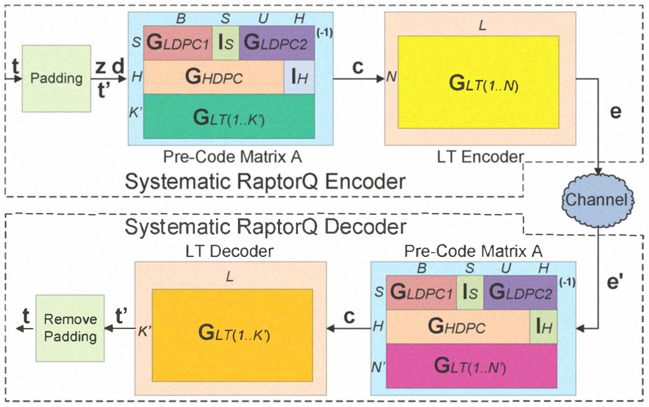

Fig. 2. Block diagram of systematic RaptorQ encoder and decoder.

The pre-code matrix $\mathbf{A}$ encapsulates several submatrices. $\mathbf{G}_{LDPC1}$ and $\mathbf{G}_{LDPC2}$ are the generator matrices of two regular low density parity check codes (LDPC), defined over Galois finite field with 2 elements GF(2). $\mathbf{G}_{HDPC}$ is the matrix of a high density parity check code (HDPC), defined over GF(256). This HDPC code matrix is the main difference that sets apart RaptorQ code from Raptor code and, to a greater extend, any other linear block code in use today. The first K' rows of the Luby Transform (LT) [7] $\mathbf{G}_{LT(1..K')}$ matrix are included into the pre-code matrix $\mathbf{A}$ to render the whole RaptorQ code systematic. $\mathbf{I}_S$ and $\mathbf{I}_H$ are identity matrices. The number of rows N ($N \geq K$) in the LT encoder matrix $\mathbf{G}_{LT(1..N)}$ are set according to the desired rate and the expected channel erasure probability. The RaptorQ encoding process is performed according to (1), where the most time consuming operation is the inversion of matrix $\mathbf{A}$.

$$\mathbf{e}_{N\times 1} = \mathbf{G}_{LT(1..N)} \cdot \mathbf{A}^{-1}_{L\times L} \cdot \mathbf{d}'_{L\times 1} = \mathbf{G}_{LT(N..1)} \cdot \mathbf{c}_{L\times 1} \quad (1)$$

The decoding process of RaptorQ code exchanges the positions of the pre-code matrix $\mathbf{A}$ and the $\mathbf{G}_{LT}$ Encoder (to be used as $\mathbf{G}_{LT}$ Decoder) matrix, as illustrated in Fig. 2 with $\mathbf{G}_{LT}$ LT generator matrices appropriately sized. This allows for successful decoding when only the first K encoded symbols have been received and no errors are detected in the channel. The input vector $\mathbf{e}'$, containing N' ($K \leq N' \leq N$) encoded symbols (which may be nonconsecutive), is padded with $S+H$ zeroes to size it to ($M = N'+S+H$). Starting with $N' = K$ the value of N' is iteratively incremented to make the matrix $\mathbf{A}$ invertible. The difference ($N' - K$) is equal to or greater than the number of received encoded symbols lost in the channel. The RaptorQ decoding process is performed according to (2).

$$\mathbf{t}'_{K'\times 1} = \mathbf{G}_{LT(1..K')} \cdot \mathbf{A}^{-1(T)}_{M\times L} \cdot \mathbf{e}'_{M\times 1} = \mathbf{G}_{LT(1..K')} \cdot \mathbf{c}_{L\times 1} \quad (2)$$

The pre-code matrix $\mathbf{A}$ in the RaptorQ encoder has to be inverted only once, for a fixed source block size K. On the other hand, the pre-code matrix $\mathbf{A}$ in the RaptorQ decoder has to be inverted with every decoded block of data.

For the purpose of comparison, the Raptor code structure and operation for application layer FEC can be consulted in [5], [8], [9].

III. RaptorQ Decoding

The specification of RaptorQ code in [3] proposes an efficient inactivation decoding gaussian elimination (IDGE) decoding algorithm. This mechanism combines the low-complexity of belief-propagation (BP) decoding with the decoding guarantee of Gaussian elimination (GE) decoding. The vector decoding is performed in parallel with the matrix inversion. The IDGE decoding algorithm achieves better decoding performance, when the pre-code matrix $\mathbf{A}$ is mapped to the memory in sparse form.

The IDGE algorithm has five phases in which it inverts the pre-code matrix $\mathbf{A}$. *Phase I* and *Phase III* use BP decoding, while *Phase II* uses GE.

In *Phase I* the algorithm seeks out the symbols that can be decoded using BP. At the beginning of *Phase I* matrix $\mathbf{A}$ is copied into an additional matrix $\mathbf{X}$ with the same dimensions. Then the algorithm looks for the row with minimum nonzero elements. If there is no row with 1 nonzero element, IDGE builds graphs of all connected rows and columns with 2 nonzero elements and chooses a row from the graph with the most components in it. In this way, it ensures that subsequently there will be the largest set of rows with just one nonzero element in them. The IDGE algorithm brings the chosen row to the position, where the main diagonal has a zero element. Then the columns are exchanged so that this zero element is replaced with a nonzero one. The columns with the remaining nonzero elements on the row are moved to the rightmost part of the matrix and excluded (inactivated) from the decoding process. A forward elimination is perform for the rows under the current one where the arithmetic operations are performed over GF(256). In order to speed up the decoding, the rows with elements over GF(256) should be used only if there are no other suitable rows with GF(2) elements.

The process repeats until *Phase I* completes successfully, when the i nonzero main diagonal elements reach the u inactivated columns. The process fails if there are zero main diagonal elements outside the part of matrix $\mathbf{A}$, containing the inactivated columns. When row and column exchange operations and row division operation are performed on matrix $\mathbf{A}$, the same operations are performed on matrix $\mathbf{X}$, too.

However, no row XOR operations are performed on matrix $\mathbf{X}$. At the end of this phase, matrix $\mathbf{A}$ is reduced to the form shown in (3). Sub-matrix I_i is an identity matrix, sub-matrix $Z_{(n-i)\times i}$ is a zero matrix, and sub-matrix $U_{n\times u}$ is a nonzero matrix, which will be processed in the remaining phases of the IDGE algorithm.

$$\mathbf{A}_{Phase_I(n\times m)} = \left[\begin{array}{c|c} \mathbf{I}_i & \\ \hline \mathbf{Z}_{(n-i)\times i} & \mathbf{U}_{n\times u} \end{array}\right] \tag{3}$$

Phase II of the IDGE algorithm converts the lower part of the sub-matrix $U_{n\times u}$ to an identity matrix I_u, through the standard GE process. At this point the success or failure of the GE process determines whether the whole IDGE decoding is successful or not. At the end of a successful *Phase II*, matrix $\mathbf{A}$ becomes square, as shown in (4). In *Phase II* all the entries of matrix $\mathbf{X}$ outside the first i columns and rows are discarded so that the matrix has a lower triangular form.

$$A_{Phase_{II}} = \begin{bmatrix} \mathbf{I}_i & \mathbf{U}_{i\times u} \\ \mathbf{Z}_{u\times i} & \mathbf{I}_u \end{bmatrix} \tag{4}$$

At the beginning of *Phase III* the part of matrix $\mathbf{A}$ that remains to be zeroed is $U_{i\times u}$ in (4). This submatrix is typically dense. To reduce it to a sparse form, matrix $\mathbf{X}$ is multiplied with the submatrix of the first i rows of the pre-code matrix $\mathbf{A}$. After this operation the first i rows and columns of matrix $\mathbf{A}$ have lower triangular form and are equal to matrix $\mathbf{X}$, while $U_{i\times u}$ is sparse.

$$A_{Phase_V} = \begin{bmatrix} \mathbf{I}_i & \mathbf{Z}_{i\times u} \\ \mathbf{Z}_{u\times i} & \mathbf{I}_u \end{bmatrix} \tag{5}$$

In *Phase IV*, matrix I_u is used to zero matrix $U_{i\times u}$. In *Phase V* forward elimination is used to zero the first i rows of matrix $\mathbf{A}$ (5). The process is guaranteed to be successful, because all the main diagonal elements are different than zero.

IV. Experimental Results

The coding performance of block codes, such as Raptor and RaptorQ codes, over an arbitrary GF(q) field has a theoretical limit. At the receiver side, the approximate probability of successful decoding with $O = 1, 2, 3, ...$ extra symbols (in addition to any K correctly received encoded symbols) is given by (6) [10]. Fig. 3 plots the possible failure probability of a block code over GF(q). It is easily noted that large size finite fields bring a tremendous performance improvement. The coding performance of RaptorQ code benefits from that fact.

$$\mathbf{P_{success}} \approx 1 - \frac{1}{q^{O+1}} \tag{6}$$

Our simulation results showed that RaptorQ provides a 99% probability of successful decoding with no extra symbols, while Raptor code requires 8 extra symbols for the same performance. For probability of successful decoding of 99.9%, RaptorQ code requires one extra symbol, while Raptor code needs 16.

Nevertheless, this outstanding performance comes at a price: steep rise in the decoding complexity. Fig. 4 shows the RaptorQ and Raptor codes decoding times with the IDGE

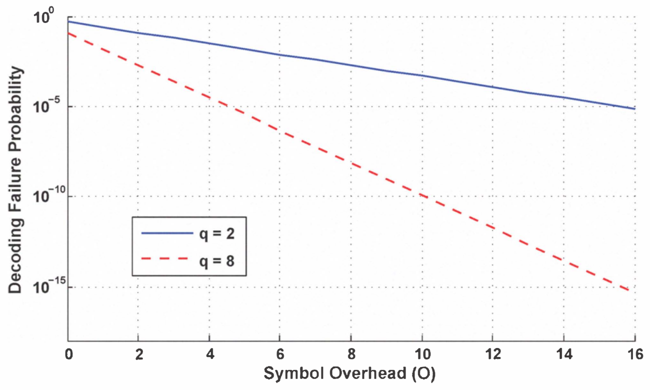

Fig. 3. Theoretical failure probability of codes over GF(q) finite fields.

algorithms specified in [3] and [4], respectively, for various block sizes K and symbol sizes T. The pre-code matrix is mapped to the memory of the embedded system in a sparse form, where only the nonzero elements are stored. This approach significantly reduces the memory footprint, whose effect is more significant for the case of RaptorQ code. Additionally, the performance of the codes is measured when they both guarantee the same probability of successful decoding: RaptorQ code uses no extra symbols, while Raptor code uses 8.

The plots for RaptorQ code show a constant trend and run in parallel for all but block size K values of 8 and 16 symbols. In those cases the operations on the large size symbols are dominating over the operation for the inversion of the matrix. On the other hand, for large block sizes K the inversion of the GF(256) matrix $\mathbf{A}$ dominates over the vector decoding. The RaptorQ IDGE algorithm with $T = 128$ is 2.92, 3.32 and 3.29 times faster than RaptorQ IDGE with $T = 512$, for $K = 32$, 256 and 1024, respectively. The similar values for $T = 128$ and $T = 1024$ are 6.03, 6.66 and 6.36.

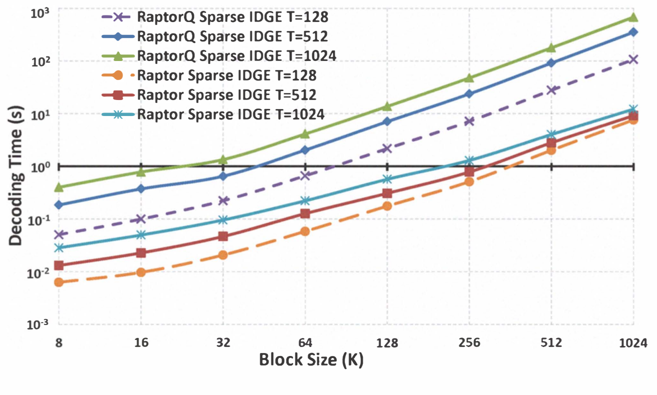

Fig. 4. Decoding time comparison between RaptorQ and Raptor codes.

The plots representing the decoding time of Raptor code converge towards the same point at large block sizes K. Opposite to the case of RaptorQ code, here the large symbol vector decoding dominates the matrix inversion for large block sizes K. The Raptor IDGE algorithm with $T = 128$ is 2.24, 1.52 and 1.2 times faster than Raptor IDGE with $T = 512$, for $K = 32$, 256 and 1024, respectively. The similar values for $T = 128$ and $T = 1024$ are 4.62, 2.53 and 1.59.

Overall, Raptor code is significantly faster than RaptorQ code. It is to be expected, based on the added GF(256) complexity. Raptor code with $T = 128$ is 10.74, 13.97 and 14.02 times faster in inverting the pre-code matrix and decoding the encoded symbol vector compared to RaptorQ code, when $K = 32$, 256 and 1024, respectively. The similar values ,when the symbol size is $T = 1024$ bytes, are 14.04, 36.8 and 55.94. With large symbols and block sizes Raptor code is overwhelmingly faster than RaptorQ code.

The work in [6] demonstrated the FEC decoding time of Raptor code for an MBMS system with bearer speed of 384 kbps and an erasure rate of 20%. Raptor code with IDGE decoding algorithm is able to fit into the transmission time slot and decode the data for most of the data block sizes ($K \times T$ in KB).

Fig. 5 shows the similar scenario, when RaptorQ code is employed with its IDGE decoding algorithm. The values above the bars show the time remaining, before the decoding of the next received block of data must start, in order to keep real time multimedia broadcasting. The negative values in Fig. 5 show that RaptorQ is not able to perform the FEC on time with any of the K and T combinations. In the best case of $K = T = 128$ it is still 1.62 seconds short of decoding time.

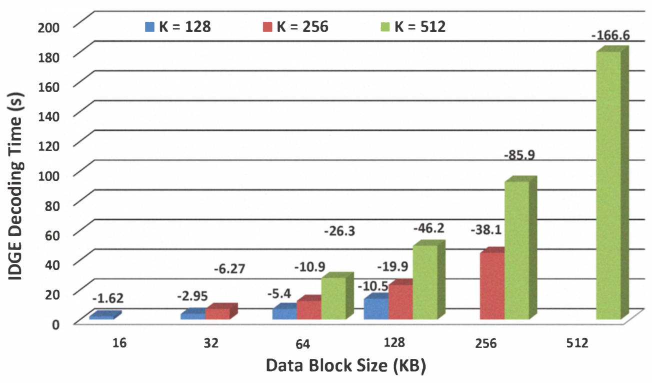

Fig. 5. MBMS FEC decoding with RaptorQ code and the IDGE algorithm.

The good coding properties of RaptorQ code can be used for data delivery, where the decoding speed comes second to the reduction of the transmitted redundant data. For real time application, a FEC system employing RaptorQ would require a much higher processor clock frequency and a hardware acceleration, similar to the one reported in [11] for Raptor code.

TABLE I
POWER AND ENERGY COMPARISON BETWEEN RAPTORQ AND RAPTOR CODE ON EMBEDDED SYSTEM.

	Sparse RaptorQ IDGE			**Sparse Raptor IDGE**		
T (bytes)	**128**	**512**	**1024**	**128**	**512**	**1024**
Power [mW]	1,352			1,354		
Energy [J] K=128	2.94	9.54	18.54	0.24	0.42	0.76
Energy [J] K=256	9.66	32.1	64.42	0.69	1.06	1.75
Energy [J] K=512	37.68	123.5	240.4	2.73	3.79	5.44
Energy [J] K=1024	144.5	476.2	920.4	10.3	12.4	16.47

Logic Elements	6,423
Logic Registers	5,554
Memory Bits	85,152

Table I shows the power and energy profile of the software implementation of RaptorQ and Raptor codes on the chosen embedded platform. Additionally, the implementation resources are also given. As far as the two codes run in software on the same hardware platform, the power they dissipate is very similar. Nevertheless, RaptorQ consumes up to 55.86 times more energy than Raptor code with symbol size $T = 1024$ and block size $K = 1024$. This energy inefficiency is a direct consequence from the slow decoding process.

V. Conclusion

This paper focused on evaluating the new RaptorQ code on embedded system. The improved coding properties were demonstrated. The inactivation decoding Gaussian elimination algorithm was used for encoding and decoding. The decoding speed of the code, with various source block sizes and symbols sizes, was compared to that that of Raptor code. A real time MBMS system scenario was used to asses the suitability of RaptorQ code for multimedia broadcasting on a mobile embedded platform. Additionally, the energy efficiency of Raptor and RaptorQ codes were demonstrated and compared. It was found that although the RaptorQ code has excellent coding properties, it is complex for efficient real time decoding for MBMS like systems.

Acknowledgment

This work was supported in parts by the National IT Industry Promotion Agency of Korea, in part by the Center for Distributed Sensor Network at GIST, and in part by the World Class University (WCU) program at GIST, through a grant provided by the Ministry of Education, Science and Technology (MEST) of Korea (Project No.R31-2008-000-10026-0).

References

[1] *3GPP TS 26.346, Techn. Spec. Group Serv. and Sys. Aspects; Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs*, 3GPP Techn. Spec., Rev. V7.4.1, June 2007.

[2] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, pp. 2551–2567, June 2006.

[3] *Reliable Multicast Transport: RaptorQ Forward Error Correction Scheme for Object Delivery*, IETF Internet Draft, Feb. 2011.

[4] *IETF RFC 5053 (2007): Raptor Forward Error Correction Scheme for Object Delivery*, IETF Proposed Standard, Sep. 2007.

[5] T. Mladenov, S. Nooshabadi, and K. Kim, "Implementation and evaluation of Raptor codes on embedded systems," *IEEE Trans. on Comp.*, 2010.

[6] ——, "MBMS Raptor codes design trade-offs for IPTV," *IEEE Trans. on Cons. Elect.*, vol. 56, no. 3, pp. 1264 –1269, Aug 2010.

[7] M. Luby, "LT codes," in *Annual IEEE Symp. on Found. of Comp. Sci.*, Nov. 2002, pp. 271–280.

[8] M. Luby, T. Gasiba, T. Stockhammer, and M. Watson, "Reliable multimedia download delivery in cellular broadcast networks," *IEEE Trans. on Broadcasting*, vol. 53, no. 1, pp. 235–246, Mar. 2007.

[9] T. Gasiba, T. Stockhammer, and W. Xu, "Reliable and efficient download delivery with raptor codes," in *Proceedings, Fourth International Symposium on Turbo Codes & Related Topics, Munich*, Apr. 2006.

[10] J. Blmer, R. Karp, and E. Welzl, "The rank of sparse random matrices over finite fields," *Random Structures & Algorithms*, vol. 10, no. 4, pp. 407 – 419, 1997.

[11] T. Mladenov, S. Nooshabadi, K. Kim, and A. Dassatti, "Parallel scalable hardware architecture for hard Raptor decoder," in *Proc., IEEE Int. Symp. on Cir. and Sys. (ISCAS)*, May 2010, pp. 3741 –3744.