Analysis and Implementation of Raptor Codes on Embedded Systems

T. Mladenov and K. Kim
Dept. Info. & Commu.,
Gwangju Inst. of Sci. and Tech.,
Gwangju 500-712, South Korea,
Email: {todor,kskim}@gist.ac.kr,

S. Nooshabadi
Dept. of EE & CE,
Michigan Tech. Uni.,
Houghton, MI 49931
Email: saeid@mtu.edu

A. Dassatti
VLSI Lab, Electronic Dept.,
Politecnico di Torino,
10130 Torino, Italy,
Email: alberto.dassatti@polito.it

***Abstract*—Raptor codes have been proven very suitable for mobile multimedia content delivery, and yet they have not been analyzed in the context of embedded systems. At the heart of Raptor codes for binary erasure channel (BEC) is the matrix inversion operation. This paper analyzes the performance, energy profile and resource implication of two matrix inversion algorithms for the Raptor decoder on a system on a chip (SoC) platform with a soft-core embedded processor. We show how the cache size, matrix memory type and organization affect the two algorithms under consideration.**

I. Introduction

Raptor codes have drawn significant attention since their introduction in [1]. They come as a powerful extension of Luby transform (LT) codes [2], [3], providing linear time encoding (regardless of the quantity of repair data generated), linear time decoding (independent of the amount of loss) and very close to the ideal code performance under any channel loss condition.

Multimedia on mobile devices requires delivery of various sized data with minimum negotiation overhead. Raptor codes have come quite useful for forward error correction (FEC) on BEC and outperformed the already well known coding schemes. Recently there have been two standards, namely 3GPP MBMS (Multimedia Broadcast/Multicast Services) [4] and DVB-H [5], which have included systematic Raptor codes in their specifications for content delivery.

The operation of Raptor codes, with some guidelines for their implementation to achieve good performance, as specified in 3GPP, can be found in [3]. Implementation and performance evaluation of Raptor codes for multimedia applications, on a workstation platform, are discussed in [6]. As far as we are aware, their software implementations on a mobile embedded system have not yet been investigated.

This paper looks at the implementation of Raptor codes on an embedded system platform, where resources in terms of the CPU speed, computational power, memory, and power dissipation are limited. We investigate the matrix inversion operation, the most demanding part of the Raptor decoder, by implementing it using two algorithms; the well known Gaussian elimination (GE) and the efficient matrix inversion algorithm (SA) proposed in 3GPP [4], and DVB [5] standards. We investigate the relative performance of these two algorithms in terms of decoding time, power and energy on an embedded processor platform. The effects of different memory types and organization for the matrix storage, the cache size, the performance behavior of the Raptor codes are demonstrated for the software implementation of both algorithms. Finally, based on the profiling data, suitability assessments are made for the implementation of GE and SA on an embedded system platform.

The chosen embedded system platform is a NIOS II soft-core processor, running on an Altera Stratix FPGA. Fig. 1 depicts the high level block diagram of the embedded SoC platform for the implementation of Raptor codes.

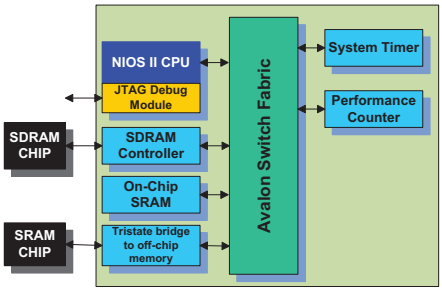

Fig. 1. Raptor codes on NIOS II embedded system.

II. Raptor Codes

Raptor codes, being a class of the Fountain codes, have the ability to generate as many encoding symbols as needed on-the-fly. The decoder can recover the source symbols from a set of slightly more encoded symbols, with a performance very close to the ideal erasure code.

A. Systematic Raptor Encoding

A block diagram of *Systematic Raptor Encoder/Decoder* is shown in Fig. 2. The encoding process is summarized in two main blocks, namely the *Code Constrains Preprocessor* and the *LT Encoder* [4], [5].

1) Code Constraints Processor: Let $\mathbf{t}$ denote K 32-bit source symbols that are to be encoded. Then $\mathbf{d}$, at the input of the Raptor encoder, contains $(L = K + H + S)$ symbols, (with H half and S parity symbols) [4], [5], and is defined as:

$$\mathbf{d}_{[0:L-1]} = [\mathbf{z}^T \quad \mathbf{t}^T]^T \tag{1}$$

978-1-4244-7773-9/10/$26.00 ©2010 IEEE

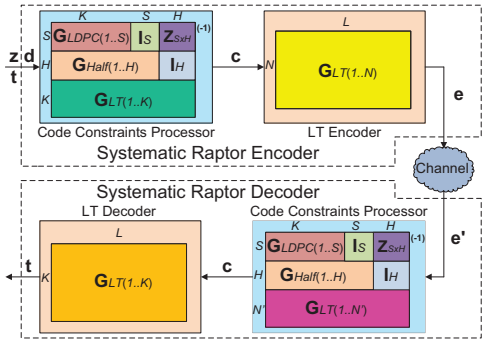

Fig. 2. Block diagram of the systematic Raptor codes.

where $\mathbf{z}_{[0:H+S-1]}$ is a vector of zeros.

The parameter S is the smallest prime integer such that:

$$S \geq \lceil (0.01 \times K) \rceil + X \tag{2}$$

with $X(X-1) \geq 2K$. Similarly the parameter H is the smallest integer such that:

$$\binom{H}{\lceil H/2 \rceil} = \frac{H!}{2(H/2)!} \geq K + S \tag{3}$$

The *Code Constrains Preprocessor* multiplies $\mathbf{d}$ with inverse pre-coding matrix $\mathbf{A^{-1}}$ to produce intermediate symbols $\mathbf{c}$:

$$\mathbf{c}_{[0:L-1]} = \mathbf{A}^{-1}_{L \times L} \cdot \mathbf{d}_{[0:L-1]} \tag{4}$$

with

$$\mathbf{A}_{L \times L} = \begin{bmatrix} \mathbf{G}_{LDPC} & \mathbf{I}_S & \mathbf{Z} \\ \mathbf{G}_{Half} & & \mathbf{I}_H \\ \mathbf{G}_{LT} & & \end{bmatrix} \tag{5}$$

where submatrices $\mathbf{I}_S$ and $\mathbf{I}_H$ are identity matrices, and $\mathbf{Z}$ is a zero submatrix of dimension $S \times H$; $\mathbf{G}_{LDPC}$ is a $S \times K$ low density parity check (LDPC) generator defined as:

$$G_{LDPC} \cdot [c[0], ..., c[K-1]]^T = (c[K], ..., c[K+S-1])^T \tag{6}$$

$\mathbf{G}_{Half}$ is a $H \times (K+S)$ matrix of Half symbols, defined as:

$$\mathbf{G}_{Half} \cdot [c[0], ..., c[S+K-1]]^T = [c[K+S], ..., c[K+S+H-1]]^T \tag{7}$$

$\mathbf{G}_{LT}$ is a $K \times L$ LT generator submatrix included in matrix $\mathbf{A}$ for the first K symbols to render Raptor codes systematic:

$$\mathbf{G}_{LT} \cdot [c[0], ..., c[L-1]]^T = [t[0], ..., t[K-1]]^T \tag{8}$$

2) LT Encoder: With source vector $\mathbf{t}$ with K symbols processed by *Code Constraints Processor*, *LT Encoder* generates any number of encoded symbols $\mathbf{e}$ according to:

$$\mathbf{G}_{LT} \cdot \mathbf{c} = \mathbf{e}_{[0:N-1]} \tag{9}$$

where $\mathbf{G}_{LT}$ is an $N \times L$ LT generator matrix, with $N \geq K$. The value of N is selected sufficiently larger than K to compensate for the possible loss of encoded symbols in the channel and, hence, to make $\mathbf{A}$ invertible at the decoder side.

Although LT itself is a nonsystematic code, the overall Raptor code is systematic:

$$e[i] = d[S+H+i], \quad \forall_{i=0,...,K-1} \tag{10}$$

That is because $\mathbf{G}_{LT}$, for symbols ($i = 0...K-1$), is included in the pre-processing matrix $\mathbf{A}$, therefore, making the resulting overall Raptor code systematic. For a fixed value of K, $\mathbf{G}_{LDPC}$, $\mathbf{G}_{Half}$ and $\mathbf{G}_{LT}(0..K-1)$ are pre generated once and stored in memory.

The structure of $\mathbf{G}_{LT}$ shows how the encoded output symbols $\mathbf{e}$ are generated from the intermediate $\mathbf{c}$ symbols. The "1" values on the g^{th} row of $\mathbf{G}_{LT}$ identify the intermediate symbols that are XORed to generate the encoded symbol $e[g]$.

Before the encoded symbols are sent to the channel they are grouped and augmented with their corresponding encoding symbol ID (ESI), the details of which are omitted from the diagram in Fig. 2 in the interest of simplicity.

B. Systematic Raptor Decoding

The decoding process of Raptor codes exchanges the positions of the *Code Constraints Processor* and the *LT Encoder (Decoder)* [3], as illustrated in Fig. 2, with the $\mathbf{G}_{LT}$ LT generator matrices appropriately sized. 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 decoding is performed according to:

$$\mathbf{c}_{[0:L-1]} = [\mathbf{z}^T \quad \mathbf{e'}^T] \cdot \mathbf{A}^{-1}_{M \times L} \tag{11}$$

$$\mathbf{t}_{[0:K-1]} = \mathbf{G}_{LT} \cdot \mathbf{c}_{[0:L-1]} \tag{12}$$

where $\mathbf{G}_{LT}$ is a $K \times L$ LT generator matrix.

At the decoder side the submatrix $\mathbf{G}_{LT}(1..N')$ is first built from the input data. The ESI of the n^{th} received encoded symbol is used to generate the n^{th} row of the submatrix $\mathbf{G}_{LT}(1..N')$ through the LT encoding process.

III. Matrix Inversion Algorithms

The most common matrix inversion algorithm is GE. The main operations involved in this algorithm are "row exchange" and "row XOR" (Exclusive-OR).

The specifications for 3GPP and DVB-H standards [4], [5], recommend SA as a more efficient technique for matrix inversion. The operation of SA is divided in phases as follows.

In *Phase I* matrix $\mathbf{A}$ is reduced to the following form:

$$\mathbf{A}_{Phase_I(M \times L)} = \left[\begin{array}{c|c} \mathbf{I}_i & \\ \hline \mathbf{Z}_{(M-i) \times i} & \mathbf{U}_{M \times u} \end{array} \right] \tag{13}$$

This reduction is performed iteratively, by first relocating the rows containing the minimum number of "1s" to the top, and then moving the first column having "1" in this row to the beginning at column location i, and the remaining columns with "1" to the end of the row at column locations $m-u-1$.

Note that i and u are initialized to 0. While, in each row, i increments only once per row, u can increment multiple times.

In each iteration of the algorithm one row from the top is excluded from the consideration. Further, the count of "1s" within a row is confined to columns i to $(m-u-1)$. *Phase I* is completed when $(L = i + u)$.

In *Phase II* submatrix $\mathbf{U}$ is partitioned into lower and upper submatrices $\mathbf{U'}_{i\times u}$ and $\mathbf{U''}_{M-i\times u}$, respectively. The lower matrix $\mathbf{U''}_{M-i\times u}$ is transformed into the identity matrix $\mathbf{I}_u$ through the normal GE technique. The $(M-L)$ rows that are left below $\mathbf{I}_u$ are discarded. The form of the matrix produced at the end of *Phase II* is:

$$A_{Phase_{II}} = \begin{bmatrix} \mathbf{I}_i & \mathbf{U}_{i\times u} \\ \mathbf{Z}_{u\times i} & \mathbf{I}_u \end{bmatrix} \tag{14}$$

In *Phase III* the upper matrix U' is zeroed by XOR of its individual rows with the sufficient number of rows from the lower matrix $\mathbf{I}_u$.

$$A_{Phase_{III}} = \begin{bmatrix} \mathbf{I}_i & \mathbf{Z}_{i\times u} \\ \mathbf{Z}_{u\times i} & \mathbf{I}_u \end{bmatrix} \tag{15}$$

The code profiling on the NIOS II processor, shown in Table I, highlights the fact that the inversion of $\mathbf{A}$ is the most time critical part of the system, contributing to up to 92% of the decoding time in Raptor codes. Therefore, we have concentrated our efforts on the optimization of the inversion algorithm in the Raptor decoder.

TABLE I
RAPTOR CODE PROFILING FOR $K = 1024$.

	GE	SA
Task	**Time(%)**	**Time(%)**
Initializing Raptor Code	0.57	0.04
Generating Matrix $\mathbf{A}$	3.84	1.81
Generating Matrix $\mathbf{G}_{LT}$	0.03	0.01
Inverting Matrix $\mathbf{A}$	91.93	91.28
Other	3.63	6.87
Total	100.00	100.00

IV. SOFTWARE IMPLEMENTATION

Next we show the analysis of the software implementation of Raptor codes on the SoC platform in Fig. 1. The NIOS II soft-core processor is runs at the clock speed 100 MHz.

A. Performance

The execution time performances of the two matrix inversion algorithms GE and SA are analyzed and compared when implemented on the NIOS II processor as software modules. The experimentation was carried out for $(K = 1024)$ (a typical value for MBMS applications), symbol size of 32 bits and up to two lost encoded symbols, which leads to $\mathbf{A}$ matrix with $(S = 59)$, $(H = 13)$, adding up to $(L = 1096)$ columns. There are eight more rows than columns $(N = K + 8,\ M = 1102)$ to compensate for the two lost encoded symbols.

Two types of memory organizations for matrix $\mathbf{A}$ have been investigated. In the first organization, (denoted as "WORD") each 1-bit matrix element is assigned to a 32-bit memory word. In the second organization, (denoted as "PACKED WORD"), 32 matrix elements are packed together into a single 32-bit memory word, therefore, significantly (at best 32 times) reducing the size of the required memory.

We have investigated the effect of data cache on the performance of GE and SA inversion algorithms. Additionally, we have analyzed the effect of data memory type in the performance of the two algorithms. We have implemented the system using three types of memory for the storage of the matrix data structure; 16MB SDRAM, 1MB external SRAM, and 256KB on-chip SRAM (respectively, denoted as SDRAM, EXT_SRAM, ONCHIP_SRAM.)

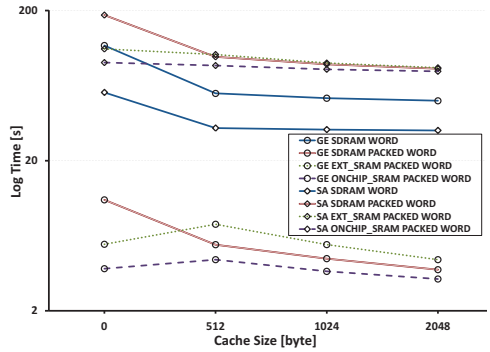

Fig. 3. *Code Constraints Processor* matrix inversion in software for K=1024.

The comparative performance of the two algorithms is summarized in Fig. 3. The sizes for the external SRAM, and 256KB on-chip SRAM memory types are not enough to fit matrix size corresponding to (K=1024) in "WORD" organization. Therefore, we report only the results for SDRAM in Fig. 3. In the absence of data cache and the "WORD" memory organization, SA performs 2.05 times better than GE for the SDRAM. These results confirm the superior performance of SA over GE, as reported in the 3GPP [4] and DVB-H [5] standard recommendations.

However, this is not the case for the "PACKED WORD". Under this memory efficient organization, GE outperforms SA by a factor of 17.03 for the SDRAM, 20.02 for the external SRAM and 23.6 for the on-chip SRAM. The difference in the values for the external and on-chip SRAMs is due to the extra read clock cycle needed by the Avalon interconnect switch (Fig. 1) while accessing the external SRAM.

The reason for the superior performance of GE over SA in the "PACKED WORD" organization can be explained as follows. SA uses "column exchange" operations in order to reduce the amount of "row exchange" and "row XOR" operations. Additionally, the counting of "1" matrix elements in Phase I of SA, significantly increases the number of memory accesses, bit masking, and bit extraction operations, which far outweigh the positive effects of the reduction in the number of "row exchange" operations. Moreover, columns are moved to the beginning or the end of the matrix, which prevents the usage of the same memory word. SA performs well under the "WORD" memory organization but lags significantly behind under the "PACKED WORD" organization.

Further, from Fig. 3 it can be seen that for the case of

"PACKED WORD" organization and zero cache size for GE, the difference in the execution time between the implementation employing SDRAM, and those employing external and on-chip SRAMs are 5.4 and 7.1 seconds. The equivalent differences for the case of SA are 75.6 and 96.5 seconds, respectively, which are far more than those for GE. This indicates that the SA benefits more significantly from a faster memory type. This is due to the increased memory accesses for the reason explained in the previous paragraph.

Fig. 3 also reveals that the effect of cache is most significant for SDRAM. The improvement is highest for the cache size of 512 bits and becomes insignificant for values greater than 1024 bits. The 512-bit cache size has negative (slowing) effect on external and on-chip SRAMs for GE, and even causes the SDRAM case to perform faster than external SRAM. That is because the cache size is not enough to hold the required amount of data, resulting in frequent cache updates. The higher rate of cache conflict, and lower cache penalty for the external and on-chip SRAMs, compared with much larger penalty incurred in the case of SDRAM, depredates the performance for the two SRAMs in comparison with SDRAM.

TABLE II
POWER, ENERGY AND HARDWARE RESOURCE FOR THE SOFTWARE IMPLEMENTATION, FOR K=128.

		No Cache			Cache 1024 Bytes		
		SDRAM	EXT SRAM	ONCHIP SRAM	SDRAM	EXT RAM	ONCHIP SRAM
Word GE	**Power [mW]**	1,864.26	1,754.18	1,881.49	1,897.68	1,781.45	1,923.58
Word SA	**Power [mW]**	1,869.61	1,743.51	1,883.09	1,898.59	1,777.79	1,925.51
Bit GE	**Power [mW]**	1,880.36	1,742.47	1,882.73	1,896.85	1,758.78	1,925.90
Bit SA	**Power [mW]**	1,871.97	1,722.72	1,881.12	1,783.55	1,697.73	1,924.94
Word GE	**Energy [mJ]**	1,522.10	568.26	429.23	1,297.97	496.85	385.26
Word SA	**Energy [mJ]**	995.01	412.72	326.03	1,131.58	417.92	316.69
Bit GE	**Energy [mJ]**	184.81	72.03	56.92	166.08	66.95	54.68
Bit SA	**Energy [mJ]**	1,711.29	750.91	660.72	955.36	603.82	612.99
Hardware	**Logic Elements**	5,130	5,333	5,264	5,628	5,745	5,702
Resources	**Memory Bits**	47,360	47,360	2,144,512	56,384	56,384	2,153,536

B. Power and Area

Table II shows the power and energy estimation for the matrix inversion operation, using SA and GE, implemented in SDRAM, external and on-chip SRAMs, for 1024 bits cache and no cache cases. The power and energy are calculated for the case of (K=128), where the matrix fits in all memories with both "WORD" and "PACKED WORD" organizations.

The total power includes the core dynamic and static and the I/O powers of the FPGA, programmed with the soft-core NIOS II processor that executes the codes for SA and GE algorithms, for the inversion of the *Code Constraints Processor* matrix. It also includes the power from the external memory chips. For a more accurate estimation of power and energy the switching activity for each case is extracted from the actual code execution profiling, and supplied to the power analysis tool. The code profiling data, along with information provided from the memory chips manufactures, were used to estimate the contribution of the power from the external memories.

According to Table II, the power dissipation does not vary significantly between the two different algorithms and memory organizations, as long as the same memory type is used. There is a difference between the three distinct memory types. The cache adds relatively small amount of additional power dissipation, although providing significant performance improvement.

The energy required to execute the codes for the two matrix inversion algorithms on the NIOS II platform is estimated as the product of execution time and the power dissipation. The GE algorithm under "PACKED WORD" memory organization shows an overwhelming better energy dissipation compared to the other cases, for all three memory types considered.

From Table II it also can be seen that the the cache memory marginally improves the energy consumption of both algorithms; a counter intuitive result. This indicates that a small power increase, due to the inclusion of the cache, is marginally outweighed by the reduction in the execution time.

Further, Table II shows the hardware resource requirements in terms of logic elements and the memory bits used by the six different implementations on FPGA. The addition of cache results in a modest increase in the amount of logic elements.

V. CONCLUSION

This paper evaluated the performance of two matrix inversion algorithms for Raptor decoder on an embedded system. Two memory organization schemes have been analyzed, with clear benefits of the "PACKED WORD" organization demonstrated. The effect of storing data on different memory types have been investigated for both algorithms. It was found, under "PACKED WORD" memory organization and small symbol sizes, that contrary to the recommendation, based on the reported profiling on a workstation platform, the Gaussian elimination performs significantly better than the alternative reportedly more efficient implementation, in terms of execution speed and energy.

ACKNOWLEDGMENT

This work was supported in parts by the National IT Industry Promotion Agency of Korea, and 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] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, pp. 2551–2567, June 2006.

[2] M. Luby, "LT codes," in *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, November 2002, pp. 271–280.

[3] M. Luby, M. Watson, T. Gasiba, T. Stockhammer, and W. Xu, "Raptor codes for reliable download delivery in wireless broadcast systems," in *Proceedings, Third IEEE Consumer Communications and Networking Conference*, vol. 1, January 2006, pp. 192–197.

[4] *3GPP TS 26.346, Technical Specification Group Services and System Aspects; Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs*, 3GPP Technical Specification, Rev. V7.4.1, June 2007.

[5] *Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Content Delivery Protocols*, ETSI Technical Specification, Rev. V1.2.1, 2006.

[6] P. Cataldi, M. P. Shatarski, M. Grangetto, and E. Magli, "Implementation and performance evaluation of LT and raptor codes for multimedia applications," in *Proceedings, IEEE International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIH-MSP'06*, 2006.