Proceedings of the 12th International Conference
on Computational and Mathematical Methods
in Science and Engineering, CMMSE 2012
July, 2-5, 2012.

Parallelization of the interpolation process in the Koetter-Vardy soft-decision list decoding algorithm

M.ª de los Ángeles Simarro-Haro[1], José Moreira[2], Marcel Fernández[2], Miguel Soriano[2], A. González[1] and F. J. Martínez-Zaldívar[1]

[1] *Instituto de Telecomunicaciones y Aplicaciones Multimedia (iTEAM) , Universitat Politècnica de València, Spain*

[2] *Departament d'Enginyeria Telemàtica, Universitat Politècnica de Catalunya, Spain*

emails: mdesiha@teleco.upv.es, jose.moreira@entel.upc.edu, marcel@entel.upc.edu, soriano@entel.upc.edu, agonzal@dcom.upv.es, fjmartin@dcom.upv.es

Abstract

List decoding is a decoding strategy that provides a set of codewords at the output of the channel decoder. Since this technique corrects errors beyond the correcting bound of the code, upper layers in the application or in the communications protocol can choose the appropriate candidate codeword among the elements of the set. The Koetter-Vardy algorithm is a soft-decision decoding algorithm for Reed-Solomon codes. It is based on two sequential processes: interpolation and factorization. In most applications it is interesting to efficiently decode in real time. This paper discusses some parallelization results about the interpolation process, which is the highest time-consuming part of the Koetter-Vardy algorithm.

Key words: Reed-Solomon, list decoding, Koetter-Vardy, Guruswami-Sudan, parallel computing, multicore, error-control coding, watermarking, fingerprinting

1 Introduction

Reed-Solomon (RS) codes are a family of error-correcting codes which have been used in many applications since their formulation. Countless applications can be found in error control coding of digital communications standards such as Digital Video Broadcasting (DVB) [9], WIMAX [8], etc. Other applications such as fingerprinting or watermarking

benefit from this family of codes in order to protect hidden information that can identify dishonest users performing unauthorized redistribution of digital contents [15, 6].

Decoding of RS codes has been an active research topic for the past decades. The last breakthrough came with the work of Sudan in list decoding [2], where the first polynomial-time list decoding algorithm for RS codes was presented. List decoding was introduced by Elias in [5]. As its name suggests, it provides at the output of the decoder not a single codeword but a set of them, in which the sent codeword can be found. This initial work was later extended by Guruswami and Sudan in [1]. Furthermore, in [3] Koetter and Vardy use reliability (soft) information provided by the channel in order to improve the decoding process.

1.1 Objectives and paper organization

Parallel implementations of the Koetter-Vardy soft-decision decoding algorithm (KV algorithm) can be found in the literature, mainly as hardware implementation on FPGAs (see [11], [12], [13]). Unfortunately, there is lack of implementations and variations in the context of parallel multicore systems. These systems are having a widespread use in handheld and mobile devices, so an efficient parallel implementation based in multithread programming is of increasing interest.

In this paper we discuss an efficient implementation of the interpolation process, which is the highest complexity part of the KV algorithm. After describing its sequential implementation, we proceed to present the parallelization scheme and its performance evaluation. Finally we give some conclusions.

2 The Koetter-Vardy soft-decision decoding algorithm

Before discussing the KV algorithm, let us define formally a Reed-Solomon code.

Let $GF(q)$ be the finite field of q elements, and let γ be a primitive element of $GF(q)$. The Reed-Solomon code of length $n = q-1$ and dimension k over $GF(q)$, denoted RS$[n,k]$, is defined as the following vector subspace of $GF(q)^n$:

$$\text{RS}[n,k] = \{(p(\gamma^1), \ldots, p(\gamma^n)) : p(x) \in GF(q)[x]_{k-1}\},$$

where $GF(q)[x]_{k-1}$ is the ring of polynomials over $GF(q)$ of degree less than k.

The codeword is transmitted through a noisy channel and the decoder receives a corrupted version of it. In this work, we focus on the decoding of RS codes using the KV algorithm [3].

 ISBN:978-84-615-5392-1

2.1 Overview of the KV algorithm

In this section we briefly review the KV algorithm. For a detailed description, we refer the reader to [3].

The KV algorithm is based on the Guruswami-Sudan algorithm. Both algorithms consist of two main steps: an interpolation step and a factorization step. In addition, Koetter and Vardy included a preprocessing step in which the reliability information provided by the channel is translated into the set of interpolation constraints used in the subsequent process. The reliability information given to the decoder usually takes the form of the likelihood that a given symbol has been sent.

Before presenting the KV algorithm, we recall that if $Q(x,y) = \sum_{i,j} q_{i,j} x^i y^j$ is a bivariate polynomial with coefficients in $GF(q)$, the $(1, k-1)$-weighted degree of $Q(x,y)$ is defined as $\max\{i + (k-1)j : q_{i,j} \neq 0\}$.

The outline of the KV algorithm for a RS$[n,k]$ code is as follows:

1. *Preprocessing step.* Translate the soft information provided by the channel into a set of interpolation constraints: $S = \{(x_l, y_l, m_l)\}$, where $x_l, y_l \in GF(q)$ and m_l is a positive integer.

2. *Interpolation step.* Construct a bivariate polynomial over $GF(q)$ of minimum $(1, k-1)$-weighted degree,
$$Q(x,y) = \sum_{i=0}^{d_x} \sum_{j=0}^{d_y} a_{i,j} x^i y^j, \tag{1}$$
such that it satisfies the interpolation constraints. That is, $Q(x,y)$, has a zero in (x_l, y_l) of multiplicity m_l for every triple $(x_l, y_l, m_l) \in S$. This is equivalent to specifying the value of the function and its $m_i - 1$ symbolic derivatives in the interpolating polynomial.

3. *Factorization step.* Find all factors of $Q(x,y)$ of the form $y - p(x)$, where $p(x)$ is a polynomial of degree $k-1$ or less. The output of the algorithm is the list of codewords $\mathcal{L}$ generated from each such $p(x)$.

We can find several algorithms that solve the interpolation process (see for example [4, 14]). We have implemented the Koetter algorithm that can be found in [7, 4]. There are several algorithms for the factoring process we want to solve [4, 10, 16, 17]. In this case, the Roth-Ruckenstein algorithm has been implemented whose main background can be found in [4].

2.2 The interpolation step

Since our focus in this work is the speedup of the interpolation step when running on multicore environments, we describe in greater detail this part of the KV algorithm.

Koetter Algorithm for Interpolation

Input: Set of interpolation constraints $S = \{(x_i, y_i, m_i)\}$; maximum y-degree dy; monomial order $(1, v)$.
Output: Bivariate polynomial $Q(x, y)$ of minimum $(1, k-1)$-weighted degree that satisfies the interpolation constraints.
Initialization: $Q_j = y^j$ for $j = 0$ to dy
Process:

```
for each (x_i, y_i, m_i) ∈ S
    for (r, s) = (0, 0) to (m_i − 1, 1)
        for j = 0 to dy
            DISC: Δ_j = coeff(Q_j(x + x_i, y + y_i), x^r y^s)
        j* = argmin_(1,k−1){Q_j : j ∈ J}
        f = Q_j
        Δ = Δ_j*
            if (j ≠ j*)
                Q_j = ΔQ_j − Δ_j f
            else (j = j*)
                Q_j = (x − x_i)f
Q(x, y) = min_(1,k−1){Q_j(x, y)}
```

For a bivariate polynomial $Q(x, y) = \sum_{i,j} q_{i,j} x^i y^j$, the function coeff$(Q(x, y), x^r\ y^s)$ simply returns $q_{r,s}$. Also, the functions $\min_{(1,k-1)}$ and $\min_{(1,k-1)}$ output the minimum $(1, k-1)$-weighted degree and the index of the minimum $(1, k-1)$-weighted degree polynomial, respectively.

3 Sequential implementation

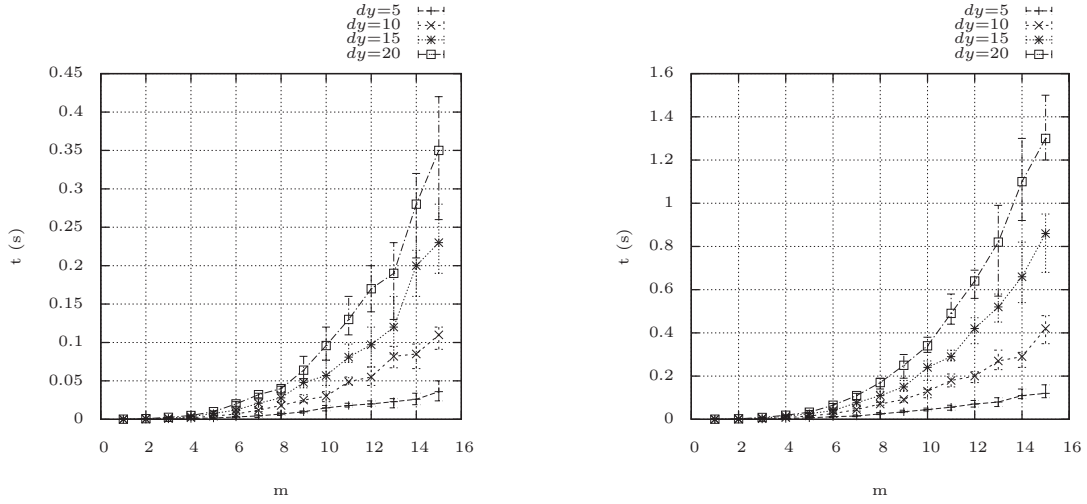
3.1 Hardware/software platform

All the tests have been executed in a SMP dual Intel(R) Xeon(TM) Hex-core CPU X5675 @3.07 GHz with 128 GB of main memory, without multithreading, running a Linux operating system with kernel 2.6.32, using the Intel `icc` C/C++ compiler version 12.1.

3.2 Execution time and serial fractions

Figures 1(a) and 1(b) show the execution time of the interpolation process versus the multiplicity m of the points for several maximum values for the grade of y (dy) and for different amount of points respectively. The measurements have been averaged 10 times, and the maximum and minimum value of the measurement are denoted in the drawn segment of it.

We can observe that the execution time grows with the multiplicity m, with the maximum grade for y, dy and with the number of points n_p.



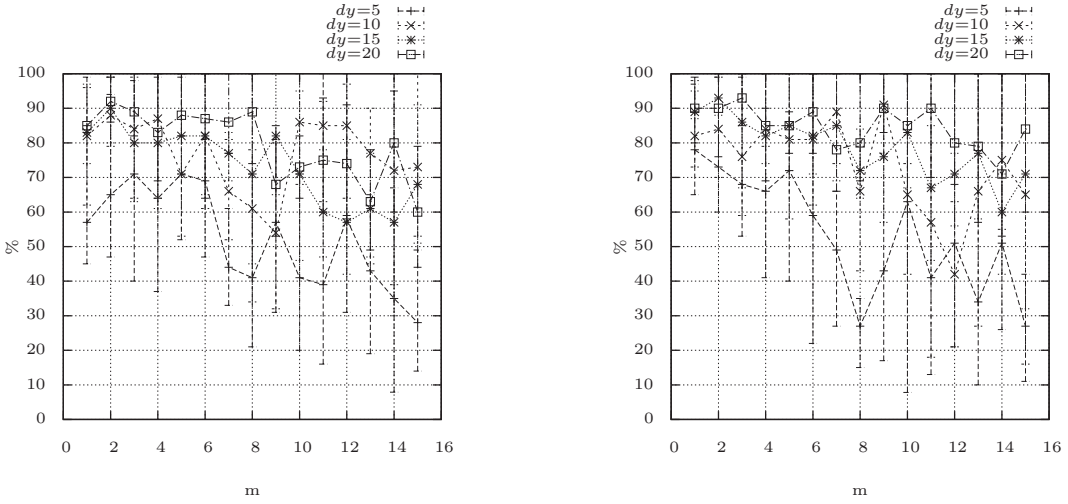
(a) $n_p = 10$ (b) $n_p = 20$

Figure 1: Sequential interpolation execution time

The most interesting information comes from the proportion of time the interpolation process takes respect to the total decoding time (interpolation time plus factorization time); this is shown in Figures 2(a) and 2(b). A great percentage of the execution time of the GS algorithm is concentrated in the interpolation part as curves show. In general, the higher value for dy, the higher proportion of interpolation execution time. The variance in the time proportion measurements is high due to a strong dependency on the problem instance. So from an efficiency point of view, it is interesting to concentrate the effort in parallelizing this part of the algorithm.

Inside the interpolation process, any iteration of the Koetter algorithm has data dependency from the previous iteration so a parallelization of any of the outermost `for` loops would be quite inefficient. In the innermost j-loop, where the *discrepancies* are computed, each iteration is completely independent from any other one, so we can parallelize it effectively. Fortunately, the rest of the code is relatively light respect to this innermost loop. Figures 3(a) and 3(b) show the ratio between the discrepancy computation execution time respect to the total interpolation execution time. This proportion is relatively constant (between 75 and 85%) with the multiplicity m and the number of points n_p and higher with higher values of d_y.

Again, this part of the code is the most time-consuming part, so its parallelization can



(a) $n_p = 10$ (b) $n_p = 20$

Figure 2: Proportion of interpolation execution time vs. total decoding execution time

potentially provide a favorable speedup in the computation. Anyway, these fractions or proportions of the parallelizable code will limit the maximum achievable speedup to the range 4–6.6 due to Amdahl's law.

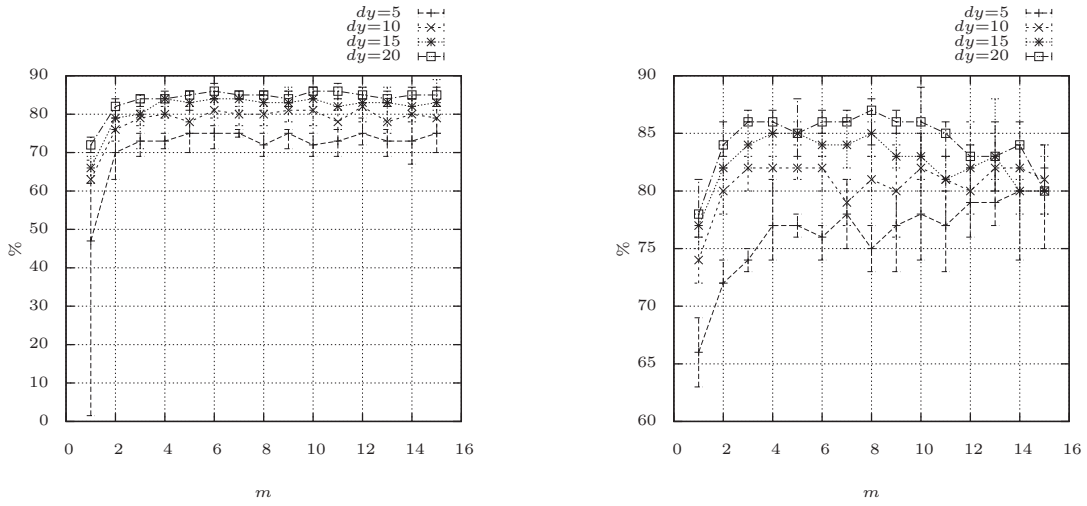
4 Parallelization and speedup

We have used *OpenMP* compiler directives to parallelize the innermost loop of the Koetter algorithm. Parallel and sequential times have been obtained for the interpolation process using the `-O3` compiler switch for all cases.

Figures 4(a) and 4(b) show the speedup in the interpolation code for two different number of points. The speedup obtained ranges between 1.7 and 3.5 depending on the parameter values. If we compare these values to the Amdahl's law maximum attainable speedup, the efficiency is in the range 43–53%.

5 Conclusions

In this paper we have shown the performance results of the interpolation process parallelization in the Koetter-Vardy algorithm. In this algorithm we find nested loops with data dependency iterations which result in a difficult parallelization. Only part of the algorithm



(a) $n_p = 10$ (b) $n_p = 20$

Figure 3: Proportion of discrepancy execution time vs total interpolation execution time

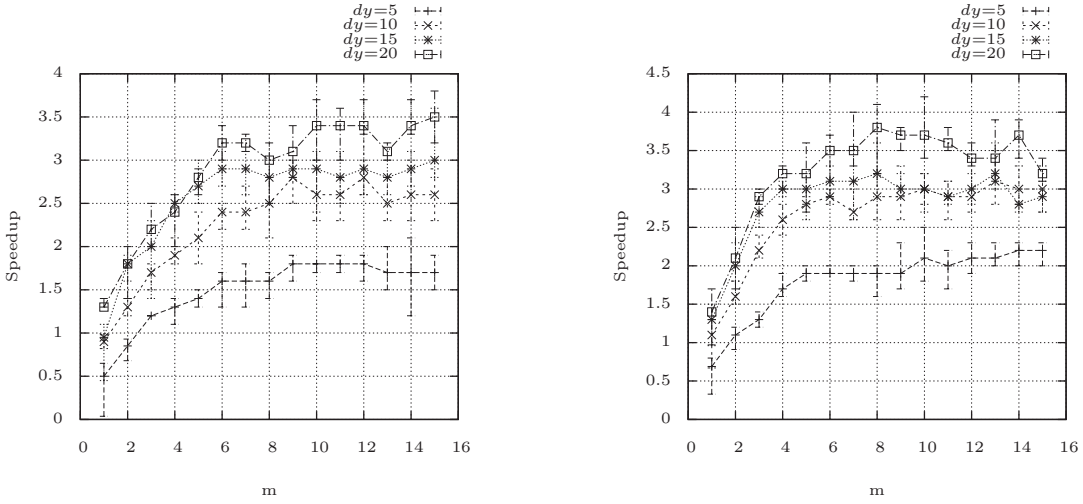
can be fully parallelized, so the serial fraction of the algorithm is meaningful and it imposes a limit in the maximum attainable speedup. This is the main reason why only a moderate speedup or parallel efficiency has been obtained.

Acknowledgements

This work was financially supported by the Vicerrectorado de Investigación de la UPV through Programa de Apoyo a la Investigación y Desarrollo (PAID-05-11-2733), Generalitat Valenciana through project PROMETEO/2009/013, by the Spanish Government through projects Consolider Ingenio 2010 CSD2007-00004 "ARES" and TEC2011-26491 "COPPI", and by the Catalan Government under Grant 2009 SGR-1362. J. Moreira is the recipient of an FPU fellowship, AP2009-3854, from the Spanish Ministry of Education.

References

[1] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon codes and algebraic geometric codes", *IEE Transactions on Information Theory*, vol. 45, no. 6 pp. 1757-1767, 1999.



(a) $n_p = 10$ (b) $n_p = 20$

Figure 4: Speedup in Koetter interpolation algorithm

[2] M. Sudan, "Decoding of Reed-Solomon codes beyond the error correction bound", *J. Complexity*, vol 12, pp. 180-193, 1997.

[3] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of Reed-Solomon codes", *IEE Transactions on Information Theory*, vol. 49, no. 11, pp. 2809-2825, 1999.

[4] Tood K. Moon, "Alternate Decoding Algorithms for Reed-Solomon Codes" *Wiley*, 2005

[5] Peter Elias, "List Decoding for noisy channels" *Technical report*, Laboratory of electronics Massachusets Institute of Technology, September 1957.

[6] M.Fernández and M. Soriano, "Fingerprinting concatenated codes with efficient identification" *Lecture Notes in Computer Science*, 2433: 459-470, 2002.

[7] R.J McEllice, "The Guruswami-Sudan Decoding Algorithm for Reed-Solomon Codes" *Technical report*, California Institute of Technology, Pasadena, California, May 2003

[8] S. Lujan, S. Almagro, A. Cabrera, J. Suardiaz and F. Cerdan "Códigos RS y su aplicación a la capa física 802.16 en FPGAs" *Universidad Politécnica de Cataluña*

[9] M. Francis and R. Green "Forward Error Correction in Digital Television Broadcast System" *Xilinx v1.0* September 5, 2007.

[10] Daniel Augot and Lancelot Pequet "An alternative to Factorization: a Speed for Sudan's decoding algorithm and its generalization to algebraic-geometric codes" *Intitut National de Recherche en Informatique et en Automatique* October 1998 INRIA no. 3532

[11] Laurier Bouulianne and Warrer J. Gross "SIMD Implementation of Interpolation in Algebraic Soft-Decision Reed-Solomon Decoding" *Signal Processing Systems Design and Implementation* pp.750-755, November 2005.

[12] Warren J. Gross, Frank R. Kschischang and P. Glenn Gulak "Architecture and Implementation of an Interpolation Processor for Soft-Decision Reed-Solomon Decoding" *IEEE Transactions on very large scale integration (VLSI) Systems*, Vol. 15, No. 3, March 2007.

[13] Bainan Chen and Xinmiao Zhang " FPGA implementation of a factorization processor for soft-decision reed-solomon decoding" *Circuits and Systems*, pp. 944 - 947, May 2008.

[14] Kwankyu Lee and Michael E. O'Sullivan " An Interpolation Algorithm using Göbner Bases for Soft-Decision Decoding of Reed-Solomon Codes" *Information Theory*, pp. 2032 - 2036, July 2006.

[15] A. Barg, G. R. Blakley and G. A. Kabatiansky "Digital fingerprinting codes: Problem statements, constructions, identification of traitors" *IEEE Trans. Inf. Theory*, pp. 852–865, (49) 2003

[16] X. W. Wu and P. H. Siegel "Efficient root-finding algorithm with application to list decoding of algebraic geometric codes" *IEEE Trans. Inform. Theory*, Vol. 47, No. 9, (September 2001), pp. 2579-2587.

[17] R. Nielsen and T. Høholdt "Decoding Reed-Solomon codes beyond half the minimum distance" *Coding Theory, Cryptography and Related Areas*, Eds. Berlin, Germany: Springer-Verlag, 2000, pp. 221-236.