UNIVERSIDAD POLITECNICA DE VALENCIA

DEPARTAMENTO DE COMUNICACIONES

Máster Universitario en Tecnologías, Sistemas y Redes de Comunicaciones

Estudio e Implementación eficiente de la decodificación de lista para códigos Reed-Solomon

Autor: Mª Ángeles Simarro Haro

Director 1: Francisco José Martínez Zaldívar

Fecha de comienzo: 1/09/2011

Lugar de trabajo: Grupo de Tratamiento de Audio y Comunicaciones del iTEAM

Objetivos – El objetivo principal del presente trabajo es el estudio e implementación eficiente de la decodificación de lista para códigos Reed-Solomon. La finalidad práctica de esta implementación es realizar la parte de decodificación de un código de *fingerprinting* propuesto en [42] por M. Fernández y M. Soriano. Para lograr el objetivo propuesto es necesario realizar un estudio teórico exhaustivo. En este estudio se analizan los pasos a seguir en la decodificación y los diferentes algoritmos que dan solución a las principales etapas del decodificador, así como los conceptos teóricos necesarios que dan lugar a su formulación. Tras el estudio realizado se pretende llevar a cabo la implementación de un software capaz de realizar la decodificación de forma eficiente en cuanto a tiempo de ejecución. Posteriormente y como parte destacable del proyecto el objetivo es el estudio e implementación de una paralelización del algoritmo de decodificación, lo cual ha dado lugar a una nueva implementación práctica de este. Por otro lado, el resultado del software realizado es embebido en un proyecto desarrollado en la Universidad Politécnica de Cataluña, cuyo objeto es una aplicación de fingerprinting para la detección de usuarios confabuladores.

Metodología – Se han estudiado los fundamentos de la codificación y decodificación de las técnicas de control de errores utilizadas. Se han llevado a cabo implementaciones secuenciales y paralelas de los algoritmos de decodificación de canal escogidos. Por último se ha verificado la validez de las implementaciones secuenciales y paralelas y se ha evaluado el rendimiento de todos estos algoritmos respecto al tiempo de ejecución, incremento de velocidad y eficiencia en la paralelización. Las implementaciones software se han realizado en C y se ha programado una interfaz Java-C empleando JNI. Los test han sido ejecutados en un multiprocesador SMP X5675 de Intel® con dos procesadores y seis núcleos por procesador, funcionando a una frecuencia de 3,07 GHz y con 128 GB de memoria principal.

Desarrollos teóricos realizados – El presente proyecto tiene una gran cantidad de trabajo teórico asociado. En primer lugar se realizó estudio teórico de los algoritmos que realizan la decodificación de lista y de la teoría asociada a ellos. La decodificación de los códigos Reed-Solomon es una decodificación de tipo polinómica, por tanto en el trabajo ha sido de gran importancia el estudio de teoría de polinomios en dos variables así como el de Cuerpos Finitos o de Galois [9] puesto que los coeficientes de estos polinomios pertenecen a un cuerpo finito.

Desarrollo de prototipos y trabajo de laboratorio – Una vez implementada una primera versión secuencial del algoritmo de Koetter-Vardy para la fase de interpolación dentro de la decodficación de lista de los códigos RS y su interfaz JNI para su validación en el entorno de pruebas, se procedió a la mejora de dicha implementación secuencial y posteriormente a una paralelización de la misma.

Resultados – Se ha mejorado notablemente los algoritmos secuenciales de partida y se han evaluado sus implementaciones. La evaluación ha consistido en analizar la fracción temporal de ejecución de cada una de las partes significativas de las que consta, identificando la parte potencialmente paralelizable y consecuentemente la fracción secuencial

resultante. Se ha deducido el máximo incremento de velocidad teórico que se puede obtener del algoritmo empleando la Ley de Amdahl y se ha verificado con una experimentación práctica de dicha paralelización. La paralelización se ha llevado a cabo empleando técnicas de programación de memoria compartida, concretamente OpenMP.

Líneas futuras – Tras los resultados obtenidos y a partir de las conclusiones extraídas, nos planteamos dos posibles líneas futuras complementarias de intestigación. Una primera línea futura de investigación propuesta es el estudio e implementación de técnicas de paralelización que empleen otros paradigmas de programación paralela así como la reconcepción algoritmica de algunas partes de las solución al problema para que sean más eficientes o susceptibles de obtener mejores rendimientos en la paralelización.

Por otro lado la finalidad ha sido la incorporación de esa decodifiación a un algoritmo de *fingerprinting*, como línea futura de investigación adicional se pretende utilizar este software como decodificador en un sistema MIMO que implemente detectores iterativos, contemplando la inclusión de un decodificador de canal de lista para mejorar el rendimiento en la detección.

Publicaciones –

1. M. Simarro Haro, José Moreira, Marcel Fernández, Miguel Soriano, A. González and F.J Martínez Zaldívar, "Parallelization of the interpolation process in the Koetter-Vardy soft-decision list decoding algorithm," in 12th International Conference Computational and Mathematical Methods in Science and Engineering, July 2012. La Manga (Spain).

2. M. Simarro Haro, José Moreira, Marcel Fernández, Miguel Soriano, A. González and F.J Martínez Zaldívar,"Implementación paralela del proceso de Interpolación del Algoritmo de Koetter-Vardy para decodificación de Códigos Reed-Solomon," in Jornadas de Paralelismo, September 2012,Elche (Spain). (Aceptado para su publicación).

Abstract – Reed-Solomon codes are widely used in current digital communication systems and its efficient decoding is interesting for practical applications. The mplemented decoding algorithm in this work is a list decoding strategy where the radius of decoding is increased so we can has a list o possible codewords in the output of the decoder. Guruswami-Sudan algorithm implements the list decoding for Reed-Solomon codes and Koetter-Vardy algorithm is an extension of the previous that incorporate *soft* decoding. Both are based on two sequentially processes: interpolation and factorization. The Koetter-Vardy algorithm has been implemented in ¡sequentially and then optimized and parallelized. At the end we discuss the results of the work.

Autor: Mª Ángeles Simarro Haro, email: mdesiha@teleco.upv.es
Director 1: Francisco José Martínez Zaldívar, fjmartin@dcom.upv.es
Fecha de entrega: 16-07-2012

Índice

1. Introducción

El propósito de los sistemas de comunicaciones es hacer llegar información desde un emisor hasta un receptor a través de un canal. En una tranmisión digital de información se persiguen los objetivos de fiabilidad, rapidez y seguridad. La codificación de canal se enmarca dentro del próposito de conseguir fiabilidad en la transmisión, mediante la cual se pretende proteger la información frente a degradaciones que se produzcan en el canal, añadiendo redundancia. La finalidad de la decodificación de canal es la detección y corrección de errores producidos en el canal de comunicación o en medios de grabación.

Existen distintas estrategias de uso y se puede hacer una clasificación en función de cómo se actúa frente a la detección de errores:

- Detectora: ARQ(Automatic Repeat Request), detecta los errores producidos en la transmisión.
- Correctora: FEC (Forward Error Correction), corrige los errores a partir de la información recibida.

1.1. Códigos Reed-Solomon

Los códigos Reed-Solomon (RS) son una clase de códigos que pueden pertenecer a la categoría FEC, es decir, corrigen los posibles datos alterados en el receptor y para ello utilizan dígitos de redundancia que permiten una recuperación a posteriori. Esta familia de códigos fueron desarrollaron en 1960 por Irving Reed y Gustav Solomon, tratándose de una clase de códigos correctores de errores con gran poder y utilidad, los cuales han sido una parte integral en la revolución de las telecomunicaciones del último tercio del siglo XX. Desde su descubrimiento han tenido múltiples e importantes aplicaciones que van desde la grabación en soportes magnéticos hasta las comunicaciones por satélite ([28],[13]).

La codificación de los códigos Reed-Solomon puede ser definida de dos formas distinas: en la primera definición las palabras código son la concatenación de la evaluación de un polinomio (polinomio mensaje) en varios puntos, mientras que en la segunda son los coeficientes de cierto polinomio.

El primer concepto de codificación Reed-Solomon describe la codificación de los símbolos del mensaje de k elementos viéndolos como coeficientes de un polinomio $p(x)$ de máximo grado $k-1$, donde tales coeficientes pertenecen a un cuerpo finito de orden q. Evaluando el polinomio en $n > k$ elementos distintos del cuerpo, obtenemos las palabras código. Tranmitiendo los puntos de obtenidos y mediante el uso de técnicas de interpolación en el receptor seremos capaces de recuperar el mensaje original. En el segundo concepto de codificación, en lugar de enviar la evaluación del polinomio en distintos puntos, los símbolos de codificación son vistos como los coeficientes de un polinomio de salida $c(x)$ construido mediante la multiplicación del polinomio mensaje $p(x)$ de grado máximo $k-1$ por un polinomio generador $g(x)$ de grado $n-k$.

En el receptor la decodificación para los códigos Reed-Solomon se basa principalmente en tres pasos:

- Cálculo de los síndromes de la palabra recibida.
- Determinación de la ubicación de los errores.
- Determinación del valor de los errores.

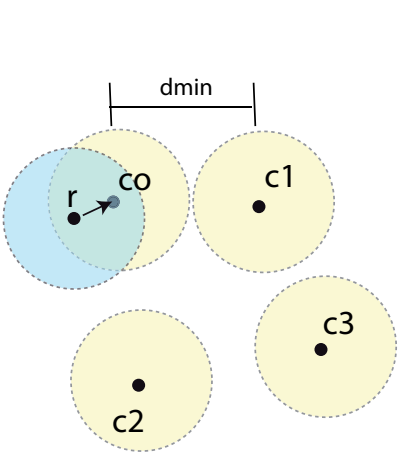
El cálculo de los síndromes es algo inmediato pues se basa en evaluar el polinomio que representa la palabra recibida en las primeras $2t$ potencias del elemento primitivo del cuerpo, obteniendo los posibles $2t$ síndromes de la palabra recibida.

La determinación de dónde están ubicados los errores se lleva a cabo calculando las raíces de un polinomio denominado "polinomio localizador de errores". El cálculo de dichas raíces se suele llevar a cabo utilizando el método de Chien. Para la determinación de los coeficientes del polinomio localizador de errores se puede emplear alternativamente el método de Peterson o bien el algoritmo de Berlekamp-Massey. Una vez identificados los símbolos recibidos que son erroneos hay que calcular el valor correcto de los mismos. Para ello podemos emplear el algoritmo de Forney. Existen también alternativas de decodificación para estos códigos RS basadas en la utilización de transformadas de Fourier en cuerpos finitos en dominios transformados.

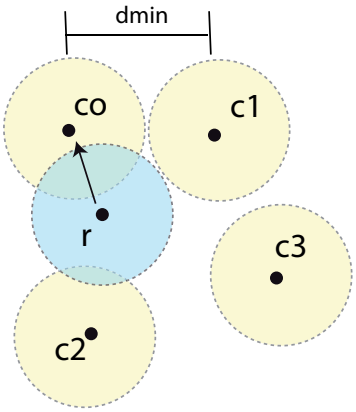
Los decodificadores mencionados arriba son decodificadores de distancia acotada, lo que significa que son capaces de decodificar hasta t_0 errores. La capacidad de correción de errores (t_0) de un código Reed-Solomon es determinada por su distancia mínima, o por $n - k + 1$. Si la ubicación de los símbolos de error no se conoce de antemano, un código Reed-Solomon puede corregir hasta $t_0 = \left\lfloor \frac{n-k}{2} \right\rfloor = \left\lfloor \frac{d_{min}-1}{2} \right\rfloor$, donde d_{min} se trata de la distancia mínima del código (distancia Hamming más pequeña entre cualquier par de palabras código). El código Reed-Solomon es un código de máxima distancia ya que es bien sabido su distancia mínima, es decir, $d_{min} = n - k + 1$ [1]. Por tanto si se producen durante la transmisión tantos errores que la distancia de la palabra recibida es mayor que t_0 no se decodificará la palabra código correcta.

Conceptualmente, esto se puede explicar de la siguiente manera. El decodificador busca en la esfera de Hamming de radio t_0 centrado en la palabra recibida (esfera de color azul en la figura 1.a). Si la esfera contiene una única palabra código, esta es la salida del decodificador. De lo contrario, el decodificador informa de fallo, llamándose éste, decodificador de distancia acotada (*bounding distance*, BD). Los algoritmos de decodificación RS convencionales son todos algoritmos BD. La esfera de decodificación no puede contener más de una palabra código, ya que la distancia mínima del código es $\geq 2t_0$. Si intentamos corregir mas de t_0 errores, al aumentar el radio de decodificación, es posible que la esfera de decodificación contenga más de una palabra código en cuyo caso el decodificador produciría un error.

[1] La distancia Hamming de un código lineal es $d \leq n - k + 1$, en el caso de los códigos Reed-Solomon la distancia mínima tiene el máximo valor posible para un código lineal, lo que se conoce como el límite Singleton.



(a) Decodificación de distancia limitada



(b) Decodificación de máxima verosimilitud

Figura 1: Tipos de decodificación única

Otro paradigma de decodificación es el decodificador de máxima verosimilitud (*maximum likelihood*, ML), también conocido como el decodificador de la palabra código más cercana, el cual es computacionalmente más difícil de conseguir. En ese caso la palabra código que retornará el decodificador a su salida será la que se encuentre más cerca de la palabra recibida. Por tanto, siempre que el número de errores e satisfaga que $e \leq t_0$, los algoritmos ML y BD decodifican exactamente la misma palabra código.

De forma convencional se afirma que un código no es capaz de corregir más de t_0 errores. Sin embargo, se puede llegar a una conclusión diferente, si barajamos la posibilidad de tener más de una palabra código a la salida. En 1950 surgió la decodificación de lista, propuesta por Elias en [10] la cual contempla la posibilidad de tener más de una palabra código a la salida del decodificador.

1.2. Decodificación por lista

En Teoría de la Codificación se suele referirse a decodificación como una decodificación única, siendo ésta una de las tareas fundamentales asociada al uso de corrección de errores, es decir, recuperar la palabra código original c_i de la palabra corrompida que se recibió r. En decodificación única, el algoritmo se ve obligado a retornar una única palabra código a partir de la palabra recibida, como se ha comentado en el apartado anterior. Avances significativos de la comunidad de la Teoría de la Información han conseguido sobrepasar esa brecha y gran parte de ese progreso se basa en el modelo de decodificación por lista.

En 1950 Elías en [10] propuso esta alternativa a la decodificación única para el caso en el cual tengamos códigos correctores de errores en canales donde se produzcan grandes tasas de errores. Al transmitir información a través de un canal ruidoso, el emisor transmite una palabra código la cual es deteriorada por ruido a través del mismo, y el receptor recibe una palabra corrupta. Si el número de errores que se producen durante la transmisión es muy grande, entonces la palabra recibida puede estar más cerca de alguna

palabra código que no sea la que se transmitió. La idea principal de decodificación por lista, como bien indica su nombre, es que el algoritmo proporciona a su salida una lista de posibilidades entre las cuales una de ellas muy probablemente es la correcta, permitiendo con ello un mayor número de errores que los permitidos en la decodificación única. Es decir, bajo este mandato el receptor produce una lista de todas las palabras código dentro de una esfera de Hamming de tamaño razonable en torno a la palabra recibida (y no solo la más cercana).

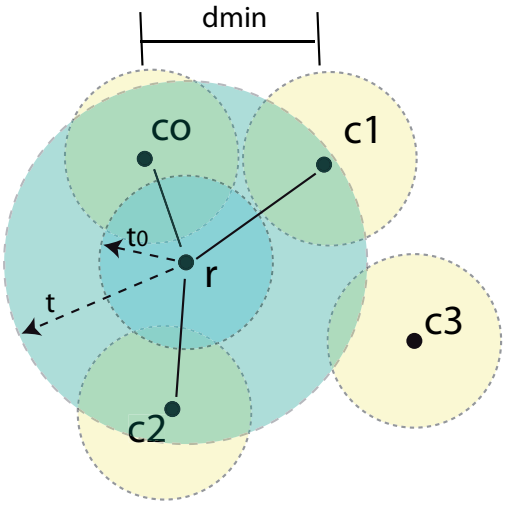


Figura 2: Decodificación por lista

Es decir en decodificación por lista el radio de decodificación es $t \geq t_0$, teniendo por tanto una esfera de decodificación mayor que la proporcionada por la distancia mínima del código. Por tanto, la cantidad de palabras código que se retornarán en la lista de salida depende del radio de decodificación. La lista de decodificación será declarada con éxito si incluye en ella la palabra transmitida. De este modo con un poco de información del contexto específico es posible "podar" la lista y recuperar la palabra código transmitida. Por lo tanto, en general, esto parece ser un modelo fuerte de recuperación de errores.

Esta noción de decodificación hasta hace poco no ha tenido algoritmos de decodificación triviales para códigos de corrección de errores. En los últimos tiempos se está estudiando algoritmos eficientes para la decodificación de lista y también su aplicación en estudios teóricos de complejidad así como en criptografía. En 1999, Madhu Sudan y Guruswami Venkatesan en el MIT publicaron "Improved Decoding of Reed-Solomon and Algebraic-Geometry Codes" [43], la introducción de un algoritmo que permite la corrección de errores más allá de la mitad de la distancia mínima del código. Este algoritmo se aplica a códigos Reed Solomon y se basa en la interpolación y la factorización de polinomios sobre $\mathbb{F}_q$ y sus extensiones.

1.3. *Fingerprinting* y*watermarking*

El algoritmo de decodificación implementado ha sido embebido como parte de un software desarrollado en la Universidad Politécnica de Cataluña. El software implementa un código de *fingerprinting* seguro propuesto en [42] y el cual está basado en la concatenación de un código externo Reed-Solomon y un código interno Simplex.

Puesto de manifiesto el fracaso de la mayoría de las técnicas anticopia empleadas para la protección de contenidos distribuibles, el *fingerprinting* o huella digital surge como una alternativa factible para luchar contra los usuarios y compradores deshonestos. Un código de *fingerprinting* es un conjunto de tuplas (palabras código) que se incorporan de forma "invisible" en cada una de las copias de un contenido que se va a distribuir a un grupo de usuarios. El proceso de incorporación de la palabra código en el contenido se realiza habitualmente a través de un algoritmo de *watermarking* o marca de agua. En caso de que dos o mas usuarios decidan realizar una confabulación para distribuir una copia ilegítima de dicho contenido(copia pirata), el código de *fingerprinting* permitirá identificarlos.

La finalidad de la inclusión de este trabajo en este proyecto es la de lograr una decodificación eficiente a efectos de coste y tiempo computacional del código Reed-Solomon de tal modo que se detecten los usuarios confabuladores de forma eficiente en tiempo de ejecución, es decir, que el proyecto pueda ser usador de forma práctica en una aplicación en tiempo real.

2. Teoría

2.1. Codificación

De los dos tipos de codificación explicados en el apartado de Introducción, la empleada para poder realizar una decodificació de lista es la que se codifica de tal modo que el mensaje son los coeficientes de cierto polinomio, con el otro tipo de codificación solo se puede emplear el paradigma de decodificación única [23].

Supongamos que tenemos un mensaje de k símbolos de información, $\{p_0, p_1, \ldots, p_{k-2}, p_{k-1}\}$ tomados del cuerpo finito $p_i \in \mathbb{F}_q$. Estos símbolos se pueden utilizar para construir un polinomio $p(x)$ del siguiente modo:

$$p(x) = p_0 + p_1 x + \cdots + p_{k-2}x^{k-2} + p_{k-1}x^{k-1} \ \in \ \mathbb{F}_q[x] \tag{1}$$

Una palabra del código Reed-Solomon se forma mediante la evaluación de $p(x)$ en n elementos del cuerpo finito $\mathbb{F}_q$.
Usando formulación matemática, tenemos la secuencia de entrada

$$P = (x_1, x_2, \ldots, x_n) \subseteq \mathbb{F}_q$$

un conjunto de $n \leq q$ elementos [2]. En ese caso el conjunto de palabras código $\mathbf{C}$ que componen el código Reed-Solomon son obtenidas mediante la evaluación de cada polinomio

[2] Generalmente $n = q - 1$ y $P = \mathbb{F}_q^* = \{0, \alpha^1, \alpha^2, \ldots, \alpha^{q-2}\}$ donde α es un elemento primitivo de $\mathbb{F}_q$.

posible $p(x) \in \mathbb{F}_q[x]$ de grado $< k$ en P:

$$C = \{(p(x_1), p(x_2), \ldots, p(x_n)) \mid p(x) \in \mathbb{F}_q[x] \wedge deg(p) < k\} \tag{2}$$

donde $\mathbb{F}_q[x]$ es el anillo de polinomios univariable sobre $\mathbb{F}_q$, y k y n son elegidas tal que $1 \leq k < n < q$.

Cuando los códigos Reed-Solomon (y cualquier otro código lineal) son referenciados, por lo general son indicados por su longitud n y por su dimensión k, referenciándose como código $RS(n, k)$. Al número de símbolos de información k, con frecuencia se le llama dimensión del código. Este término se deriva del hecho de que en el código Reed-Solomon las palabras forman un espacio vectorial de dimensión k sobre $\mathbb{F}_q$. Puesto que cada palabra código tiene n coordenadas (véase la ecuación 2), se suele decir que el código tiene una longitud de n.

2.2. Polinomios en dos variables

2.2.1. Grado y orden del monomio

Para un polinomio de una sola variable, la noción de orden es sencilla. Para un polinomio definido en múltiples variables, sin embargo, hay algún grado de flexibilidad a la hora de definir el orden del polinomio. En los algoritmos implementados se hace uso del orden y dependen de él para implementarse. En esta sección se pretenden mostrar una serie de conceptos sobre polinomios en dos variables los cuales son necesarios para el algoritmo de decodificación de lista implementado.

Un polinomio bivariable con coeficientes en $\mathbb{F}_q$ es, por definición, una suma finita de monomios de la forma

$$Q(x, y) = \sum_{i,j \geq 0} a_{i,j} x^i y^j \tag{3}$$

donde los elementos $a_{i,j} \in \mathbb{F}_q$ se denominan coeficientes y solo un número finito de ellos son no cero. Los símbolos x,y, son las indeterminadas del polinomio y se suelen representar por las letras x e y.

Si $\mathbb{F}_q$ es un cuerpo finito, se denota por $\mathbb{F}_q[x, y]$ el anillo de polinomios bivariables con coeficientes en $\mathbb{F}_q$ y se representa por

$$\mathbb{F}_q[x, y] = \{Q(x, y) : a_{i,j} \in \mathbb{F}_q\} \tag{4}$$

El sumatorio en la definición anterior del polinomio es bidimensional pero a menudo es necesario tener una representación unidimensional. Para ello necesitamos tener una ordenación del conjunto de monomios. Un monomio bivariable es una expresión de la forma

$$x^i y^j : \qquad i, j \geq 0 \tag{5}$$

Puede interpretarse que un monomio es un polinomio de un único término. En general, el valor del coeficiente de este término puede ser cualquier elemento de un cuerpo (excepto 0), no obstante, y para prescindir de la asociación del conjunto de monomios con un determinado cuerpo, tomaremos como coeficiente de los monomios el valor 1 (elemento

neutro del producto), valor que existe en cualquier cuerpo. De esta forma, el conjunto de todos los monomios bivariables se representa por

$$\mathbb{M}[x,y] = \{x^i y^j : i,j \geq 0\} \subset \mathbb{F}[x,y] \tag{6}$$

En esta sección vamos a describir una clase general de ordenación de monomios. Antes de eso notemos que el conjunto $\mathbb{M}[x,y]$ es isomórfico al conjunto $\mathbb{N}_2$ de pares no negativos de enteros, bajo la aplicación biyectiva $x^i y^j \to (i,j)$. Isomórfico se refiere a la construcción de modelos de sistemas similares al modelo original, es decir el estudio de uno de ellos puede reducirse al otro.

Un orden de monomios es una relación " $\leq$ " en $\mathbb{M}[x,y]$ (o equivalentemente en $\mathbb{N}_2$) que para cualquier $\Phi_1, \Phi_2, \Phi_3 \in \mathbb{M}[x,y]$ cumple las propiedades

1. Reflexiva, $\Phi_1 \leq \Phi_1$.
2. Antisimétrica, si $\Phi_1 \leq \Phi_2 \;\; y \;\; \Phi_2 \leq \Phi_1$ entonces implica que $\Phi_1 = \Phi_2$.
3. Transitiva, $\Phi_1 \leq \Phi_2 \;\; y \;\; \Phi_2 \leq \Phi_3$ implican $\Phi_1 \leq \Phi_3$.

Existen diversos órdenes de monomios, pero en nuestro caso, y como veremos más adelante en el desarrollo del algoritmo de Guruswami-Sudan (GS), los que nos interesan son los órdenes basados en el grado ponderado (*weighted degree*, WD). El grado ponderado es caracterizado por el par $w = (u,v)$ de enteros positivos. Por lo tanto el w-grado del monomio $x^i y^j$ es definido como

$$\deg_w x^i y^j = ui + vj \tag{7}$$

Si ordenamos el conjunto $\mathbb{M}[x,y]$ mediante un (u,v)-grado ponderado, obtendríamos solo un orden parcial, puesto que los monomios con el mismo (u,v)-grado ponderado serían incomparables.

Existen dos órdenes basados en el grado ponderado que solventan esta situación, definiendo por tanto un orden total de los monomios. Estos órdenes son el orden lexicográfico (u,v)-ponderado, abreviadamente (u,v)-lex, y el orden lexicográfico inverso (u,v)-ponderado, abreviadamente (u,v)-revlex.

El orden (u,v)-lex del conjunto $\mathbb{M}[x,y]$ establece que

$$x^{i_1} y^{j_1} \leq x^{i_2} y^{j_2} \tag{8}$$

si se cumple que

$$\begin{aligned} ui_1 + vj_1 &< ui_2 + vj_2 \qquad \text{ó} \\ ui_1 + vj_1 &= ui_2 + vj_2 \quad \wedge \quad i_1 \leq i_2. \end{aligned}$$

El orden (u,v)-revlex del conjunto $\mathbb{M}[x,y]$ establece que

$$x^{i_1} y^{j_1} \leq x^{i_2} y^{j_2} \tag{9}$$

si se cumple que

$$\begin{aligned} ui_1 + vj_1 &< ui_2 + vj_2 \qquad \text{ó} \\ ui_1 + vj_1 &= ui_2 + vj_2 \quad \wedge \quad j_1 \leq j_2. \end{aligned}$$

Utilizando cualquiera de estos dos órdenes se cumple de forma inmediata la propiedad antisimétrica de la definición de orden.

A continuación tenemos una lista ordenada de los primeros 30 monomios de $\mathbb{M}[x, y]$ según el orden lexicográfico (1, 3)-ponderado, tanto en orden revlex como lex. La primera fila de la tabla es el monomio correspondiente, bajo éste se indica el grado ponderado y en la última fila el rango del monomio.

$\Phi_j(x,y)$:	1	x	x^2	x^3	y	x^4	xy	x^5	x^2y	x^6	x^3y	y^2	x^7	x^4y	xy^2
Peso (w-revlex):	0	1	2	3	3	4	4	5	5	6	6	6	7	7	7
j:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\Phi_j(x,y)$:	x^8	x^5y	x^2y^2	x^9	x^6y	x^3y^2	y^3	x^{10}	x^7y	x^4y^2	xy^3	x^{11}	x^8y	x^5y^2	x^2y^3
Peso (w-revlex):	8	8	8	9	9	9	9	10	10	10	10	11	11	11	11
j:	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Tabla 1: Orden de los monomios bajo orden (1,3)-revlex

$\Phi_j(x,y)$:	1	x	x^2	y	x^3	xy	x^4	x^2y	x^5	y^2	x^3y	x^6	xy^2	x^4y	x^7
Peso (w-lex):	0	1	2	3	3	4	4	5	5	6	6	6	7	7	7
j:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\Phi_j(x,y)$:	x^2y^2	x^5y	x^8	y^3	x^3y^2	x^6y	x^9	xy^3	x^4y^2	x^7y	x^{10}	x^2y^3	x^5y^2	x^8y	x^{11}
Peso (w-lex):	8	8	8	9	9	9	9	10	10	10	10	11	11	11	11
j:	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Tabla 2: Orden de los monomios bajo orden (1,3)-lex

Para el desarrollo del decodificador nos será útil conocer el número de monomios que tenemos hasta un determinado grado. Para ello haremos uso del siguiente lema, cuya demostración puede encontrarse en [23].

Lema 2.1 *Sea $C(v, l)$ el número de monomios de (1,v)-grado ponderado menor o igual a l. Entonces*

$$C(v,l) = \left(\left\lfloor \frac{l}{v} \right\rfloor \right) \left(l + 1 - \frac{v}{2} \left\lfloor \frac{l}{v} \right\rfloor \right) \tag{10}$$

Una vez establecido un orden para el conjunto $\mathbb{M}[x, y]$, podemos expresar cualquier polinomio $Q(x, y) \in \mathbb{F}[x, y]$ como una suma unidimensional de monomios ponderados:

$$Q(x,y) = \sum_{j=0}^{J} a_j \Phi_j(x,y) \tag{11}$$

para algún conjunto de coeficientes $\{a_j, j = 0, \ldots, J\}$, con $a_J \neq 0$. El entero J es llamado el rango de $Q(x, y)$, denotado por $rank(Q(x, y))$, y hace referencia a la dimensión del polinomio proyectado como un vector del espacio vectorial que tiene como base los elementos de $\mathbb{M}[x, y]$. El monomio $\Phi_J(x, y)$ es llamado monomio líder de $Q(x, y)$, denotado por $LM(Q(x, y))$. El coeficiente a_J es llamado coeficiente líder de $Q(x, y)$. El grado ponderado del monomio líder de $Q(x, y)$ es llamado el grado ponderado de $Q(x, y)$, o w-grado, denotado como $\deg_w(Q(x, y))$:

$$\deg_w(Q(x, y)) = \deg_w LM(Q(x, y)) = \max\{\deg_w \Phi_j(x, y) : a_j \neq 0\} \tag{12}$$

También se dice que el y-grado de $Q(x, y)$ es el grado de $Q(1, y)$ como un polinomio en y.

2.2.2. Raíces y multiplicidad

En esta sección, continuamos con el estudio de los polinomios bivaribles, centrándonos en la noción de las raíces y su multiplicidad de estos polinomios. En el decodificador GS, estamos interesados en encontrar una interpolación polinómica con cierta multiplicidad de ceros.

Primero vamos a hacer un análisis de ceros en polinomios univariables para luego extender las conclusiones a polinomios bivariables.

Para $m \leq n$, el polinomio

$$Q(x) = a_m x^m + a_{m+1} x^{m+1} + \ldots + a_n x^n = \sum_{r=m}^{n} a_r x^r \tag{13}$$

donde $a_0 = a_1 = \cdots = a_{m-1} = 0$, se dice que tiene un cero de orden o multiplicidad m en 0. Nosotros lo denotaremos como $ord(Q; 0) = m$.

Vamos a denotar D_r como el operador derivada r-ésima. En ese caso podemos observar que

$$Q(0) = D_1 Q(0) = \cdots = D_{m-1} Q(0) = 0$$

Por ello como hemos observado el orden del cero puede ser expresado en términos de derivadas en este caso. Vamos a generalizar este resultado de ceros de orden m para otras posiciones. Decimos que $Q(x)$ tiene un cero de orden m en α si $Q(x + \alpha)$ tiene un cero de orden m en 0. Esto puede ser expresado mediante una serie de Taylor, conocida como teorema de Hasse.

Si $Q(x) = \sum_{i=0}^{n} a_i x^i \in \mathbb{F}_q[x]$, entonces para cualquier $\alpha \in \mathbb{F}_q$

$$Q(x + \alpha) = \sum_{r=0}^{n} Q_r(\alpha) x^r \tag{14}$$

donde

$$Q_r(x) = \sum_{i=0}^{n} \binom{i}{r} a_i x^{i-r}$$

y donde tomamos $\binom{i}{r} = 0$, si $r > i$.

Poniendo un pequeño ejemplo numérico comprobamos la relación anterior. Sea el polinomio $Q(x) = 2 + x - 3x^2 + x^3$ el cual posee un cero de orden 1 en $\alpha = 2$. De tal modo que $Q(x+2) = x + 3x^2 + x^3$ (pudiendo observar que $Q(x+2)$ tiene un cero de orden 1 en 0) puede ser calculado usando el teorema de Hasse del siguiente modo

$$Q(x+2) = Q_0(2) + Q_1(2)x + Q_2(2)x^2 + Q_3(2)x^3$$

donde

$$\begin{aligned} Q_0(x) &= 2 + x - 3x^2 + x^3 & \quad Q_0(2) &= 0 \\ Q_1(x) &= 1 - 6x + 3x^2 & \quad Q_1(2) &= 1 \\ Q_2(x) &= -3 + 3x & \quad Q_2(2) &= 3 \\ Q_3(x) &= 1 & \quad Q_3(2) &= 1 \end{aligned}$$

De tal modo que mediante la sustitución obtenemos el mismo resultado usando el teorema de Hasse que desplazando a la raíz α el polinomio $Q(x)$.

La prueba de lo anterior puede verse por la aplicación del teorema del binomio. $Q_r(x)$ es llamada la r-ésima derivada Hasse de $Q(x)$. Denotaremos $Q_r(x)$ por $D_rQ(x)$. En el caso en el cual $\mathbb{F}_q$ es un cuerpo de característica 0[3], entonces

$$D_rQ(x) = Q_r(x) = \frac{1}{r!}\frac{d^r}{dx^r}Q(x) \tag{15}$$

por lo que D_r actúa como un operador de diferenciación, pero con un factor de escala de $1/r!$. Podemos escribir

$$Q(x+\alpha) = \sum_{r=0}^{n} D_rQ(\alpha)x^r \tag{16}$$

Por lo tanto, si $Q(x)$ tiene un cero de orden m en α debería tener los primeros m coeficientes de la serie igual a 0:

$$Q(\alpha) = D_1Q(\alpha) = \cdots = D_{m-1}Q(\alpha) = 0 \tag{17}$$

Denotándose como $ord(Q;\alpha) = m$. Todos estos conceptos son extendidos a polinomios de dos variables. Para $Q(x,y) \in \mathbb{F}[x,y]$ y α y β siendo tales que $Q(\alpha,\beta) = 0$. Entonces diremos que Q tiene un cero en (α,β).

Con $Q(x,y) = \sum_{i,j\geq 0} a_{i,j}x^iy^j$, decimos que Q tiene un cero de multiplicidad m (u orden m) en $(0,0)$ si los coeficientes $a_{i,j} = 0$ para todo $i+j < m$. Cuando Q tiene un cero de orden m en $(0,0)$ lo denotamos como $ord(Q;0,0) = m$. De manera similar, decimos que $Q(x,y)$ tiene un cero de orden m en (α,β), denotado como $ord(Q;\alpha,\beta) = m$, si $Q(x+\alpha, y+\beta)$ tiene un cero de orden m en $(0,0)$.

[3]Cuerpo de característica 0 es aquel en el cual el 0 solo se consigue sumando 0 veces el elemento identidad, por ejemplo el cuerpo de los reales

Observamos que un cero de orden m requiere que $\binom{m+1}{2} = m(m+1)/2$ coeficientes sean 0. Por ejemplo un cero de orden 2 en $(0,0)$, los $\binom{2+1}{2} = 3$ coeficientes $a_{0,0}, a_{0,1}, a_{1,0}$ son todos cero.

Si $Q(x,y) = \sum_{i,j} a_{i,j} x^i y^j$, entonces

$$Q(x+\alpha, y+\beta) = \sum_{r,s} Q_{r,s}(\alpha,\beta) x^r y^s \tag{18}$$

donde

$$Q_{r,s}(x,y) = \sum_i \sum_j \binom{i}{r}\binom{j}{s} a_{i,j} x^{i-r} y^{j-s} \tag{19}$$

Al igual que antes la prueba es por la aplicación del teorema del binomio. Denotaremos

$$Q_{r,s}(x,y) = D_{r,s} Q(x,y) \tag{20}$$

esto a veces es llamado la derivada (r,s)-ésima de Hasse de $Q(x,y)$. La cual se calcula mediante

$$D_{r,s} Q(x,y) = Q_{r,s}(x,y) = \frac{1}{r!s!} \frac{\partial^{r+s}}{\partial x^r \partial y^s} Q(x,y) \tag{21}$$

Basándonos en esta notación, observamos que si $ord(Q;\alpha,\beta) = m$, entonces

$$D_{r,s} Q(\alpha,\beta) = 0 \quad \text{para todo } r,s \quad \text{tal que } r+s < m \tag{22}$$

que es un total de $\binom{m+1}{2}$ igualdades.

2.3. El decodificador GS: los principales teoremas

El algoritmo de Guruswami-Sudan (GS) es esencialmente un algoritmo de decodificación de lista, proporcionando una lista de todas las palabras código a distancia t_m de la palabra recibida r. La idea del algoritmo de GS se puede expresar de la siguiente manera. Bajo la construcción del código Reed-Solomon, tenemos un conjunto de puntos $(x_i, c_i),\ i = 1, \ldots, n$ los cuales son generados mediante una relación polinómica $c_i = m(x_i)$ para el polinomio $m(x)$ de grado máximo $k-1$. De modo que el conjunto de puntos (x_i, c_i) son producidos.

La palabra código c se corrompe por algún tipo de ruido al transcurrir por un canal de comunicación, o por el almacenamiento en medio digital, produciendo por tanto el conjunto de puntos (x_i, r_i). El problema ahora es adaptar un polinomio $p(x)$ de grado $< k$ a través del conjunto de puntos, tal que $p(x_i) = y_i$ (dicho de otra forma, debemos estimar cual fue el polinomio $m(x)$). Sin embargo, como existe error en algunos de los puntos, buscaremos un polinomio interpolador para cierta cantidad de puntos (el máximo posible), para $n - e$ puntos, así $|\{i : p(x_i) = y_i\}| \geq n - e$. Entonces, basado en esta interpolación polinómica, los puntos con errores son recuperados. Esto es, si i es el índice de un punto con error, entonces diremos que el valor recuperado es $\widehat{c_i} = p(x_i)$.

El decodificador Gusruswami-Sudan está generalmente basado en esta idea de interpolación. El polinomio interpolador es construido como un polinomio en dos variables con $Q(x_i, y_i) = 0$. En adición a la interpolación simple, una multiplicidad de interpolación m es introducida cuando definidos el orden de interpolación de cada punto, esta multiplicidad mejora la capacidad de corrección. Esto es más o menos equivalente a especificar el valor de la función y sus $m-1$ derivadas en el polinomio interpolador. Más adelante veremos cómo la multiplicidad puede ser usada para la decodificación *soft* del código RS, asignando a cada punto una multiplicidad m_i. Para el polinomio bivariable $Q(x, y)$ el polinomio $p(x)$ es extraído por factorización, el cual satisface que $p(x_i) = y_i$ para un número suficientemente grande de posiciones (x_i, y_i). De tal modo que cada polinomio $p(x)$ represente un posible mensaje y el conjunto de polinomios es la lista de posibles mensajes que se han transmitido. Por lo tanto hay dos principales pasos en el algoritmo de decodificación, el paso de interpolación y el de factorización.

2.3.1. Paso de interpolación

En este paso el decodificador construye un polinomio bivariable de la forma

$$Q(x, y) = \sum_{i,j} a_{i,j} x^i y^j = \sum_{j=0}^{C} a_j \Phi_j(x, y) \tag{23}$$

de grado mínimo $(1, v)$ [23], el cual tenga un cero de orden m en cada punto (x_i, y_i), $i = 1, 2, \cdots, n$. Aquí los $\Phi_j(x, y)$ son monomios de la forma $x^p\ y^q$, ordenados acorde con orden de monomios $(1, v) - revlex$ tal que $\Phi_0 < \Phi_1 < \cdots < \Phi_C$. C es conocido como el coste de interpolación, siendo el número de restricciones lineales impuestas al polinomio $Q(x, y)$, y vale

$$C = n\binom{m+1}{2} \tag{24}$$

La prueba es la siguiente. Hay un cero de multiplicidad m en (x_i, y_i) si

$$D_{r,s} Q(x_i, y_i) = 0 \quad \text{para todo} \quad (r, s) \quad \text{tal que} \quad 0 \leq r + s < m \tag{25}$$

Usando la derivada parcial de Hasse esto puede ser escrito como

$$D_{r,s} Q(x_i, y_i) = \sum_k \sum_j \binom{k}{r} \binom{j}{s} a_{k,j} x_i^{k-r} y_i^{j-s} = 0 \qquad i = 1, 2, \cdots, n, \quad \text{y} \quad r + s < m \tag{26}$$

Hay $\binom{m+1}{2}$ ecuaciones lineales homogéneas (restricciones) por cada valor de i, para tener un total de $n\binom{m+1}{2}$ ecuaciones. Si por ejemplo el valor de la multiplicidad m es 2 tendremos$\binom{2+1}{2} = 3$ ecuaciones para cada par de puntos (x_i, y_i) es decir $D_{0,0} Q(x_i, y_i) = 0$, $D_{1,0} Q(x_i, y_i) = 0$ y $D_{0,1} Q(x_i, y_i) = 0$.

Si $C = n\binom{m+1}{2}$, entonces hay $C + 1$ variables $a_0, a_1, \cdots, a_c$ en (23), garantizando que existen soluciones distinta de la trivial del sistema homogéneo

En efecto, el número de restricciones lineales impuesto por un punto de interpolación de multiplicidad m vendrá dado por el número de ecuaciones que determinan los coeficientes $a_{i,j}$, que hay un total de

$$|\{(i,j) : i + j < m\}| = \binom{m+1}{2} \tag{27}$$

ecuaciones, para cada uno de los n puntos.

Una vez descrito este paso surgen dos preguntas: ¿existe este polinomio?¿Cómo se puede construir? Como ya hemos comentado hay $C + 1$ variables $a_0, a_1, \cdots, a_C$ en (23). De ello se deduce que debe haber al menos una solución distinta de cero a este conjunto de ecuaciones, que corresponde a un polinomio $Q(x, y)$ nulo, de la forma representada en la ecuación (23) con sus correspondientes multiplicidades.

El problema puede ser resuelto utilizando álgebra lineal simple. Sin embargo, un algoritmo potencialmente más eficiente debido a Koetter es presentado en la sección 2.4. Un algoritmo capaz de lograr la solución, y el cual es una extensión del de Berlekamp-Massey a vectores es presentado en esa sección.

2.3.2. Paso de factorización

En este paso el decodificador encuentra todos los factores de $Q(x, y)$ de la forma $y - p(x)$, donde $p(x)$ es un polinomio de grado v o menos. Este paso produce una lista de polinomios

$$\mathcal{L} = \{p(x) \ \in \mathbb{F}[x] : (y - p(x))|Q(x, y)\}. \tag{28}$$

que están de acuerdo con (x_i, y_i) en al menos K_m valores, esto es, $|\{i : p_j(x_i) = y_i\}| \geq K_m$ para cada $p_j(x) \in \mathcal{L}$. Los principales resultados con respecto al paso de factorización son proporcionados por el siguiente lema y teorema de los cuales podemos encontrar sus demostraciones en [23].

Lema 2.2 *$Q(x, y)$ tiene ceros de multiplicidad m en los puntos (x_i, y_i), $i = 1, 2, \cdots, n$. Si $p(x)$ es un polinomio tal que $y_i = p(x_i)$, entonces $(x - x_i)^m|Q(x, p(x))$.*

Teorema 2.3 *Si tenemos un polinomio interpolador $Q(x, y)$ de grado ponderado $(1, v) \leq l$ tal que $D_{r,s}Q(x_i, y_i) = 0$ para $i = 1, 2, \cdots, n$, para todo $r + s \leq m$ y sin ser x_i ni y_i cero, $p(x)$ es un polinomio de grado al menos v tal que $y_i = p(x_i)$ para al menos K_m valores de i en $\{1, 2, \cdots, n\}$. De tal modo que si $mK_m > l$, entonces $(y - p(x))|Q(x, y)$.*

Las cuestiones relacionadas con este paso son: ¿Cómo se relaciona esto con la capacidad de corrección de errores del código? ¿Cómo es realizada la factorización? ¿Cuántos polinomios son incluidos en $\mathcal{L}$?

Uno de los algoritmos más eficientes para realizar la factorización es el algoritmo propuesto por Roth-Ruckenstein reduciendo ésta a una sola variable. Este algoritmo es presentado en la sección 2.5.

2.3.3. La distancia de corrección

Los parámetros m, l y t_m son función de los datos de entrada n, k y K_m. Los cuales están relacionados mediante una serie de lemas de los cuales se hará un resumen en esta sección.

Para el teorema de factorización, n es el número de puntos totales de una palabra código y K_m es el número de puntos en los que coinciden r (palabra recibida) y una palabra código, de modo que $t_m = n - K_m$ es la distancia entre r y una palabra código, siendo por tanto la distancia de corrección de errores, es decir, el radio de decodificación del diseño. A mayor valor de t_m, más errores potenciales se pueden corregir.

Ahora vamos a establecer la conexión entre la distancia t_m, la multiplicidad m y el máximo $(1, v)$-grado ponderado l de $Q(x, y)$. El principal punto del teorema de interpolación es que el número de variables en la interpolación polinómica debería exceder el número de ecuaciones, las cuales son $n\binom{m+1}{2}$. Recordar de la ecuación 10 que el número de monomios de $(1, v)$-grado ponderado l es $C(v, l)$. Por eso deberíamos tener

$$n\binom{m+1}{2} < C(v, l) \tag{29}$$

Por el teorema de factorización también deberíamos tener

$$mK_m > l \quad \text{ó} \quad mK_m \geq l + 1 \quad mK_m - 1 \geq l \tag{30}$$

Por tanto particularizando a partir de los lemas mencionados y de otros lemas y teoremas que pueden encontrarse en el artículo de Guruswami-Sudan [35] obtenemos los valores correspondientes a las variables de interés según nos muestra el algoritmo 1.

Algoritmo 1 Reconstrucción de polinomios de Guruswami-Sudan

Entrada: Dos números enteros k, K_m y n parejas de puntos $(x_i, y_i) \in \mathbb{F}_q$ con $i \in [n]$.

Salida: El conjunto de todos los polinomios $p(x) \in \mathbb{F}_q[x]$ tales que $deg(p(x)) < k$ y cumplen que $|\{i : p(x_i) = y_i\}| \geq K_m$

Paso 0: Calcular los parámetros m y l, aplicando los lemas anteriores queda

$$m \leftarrow 1 + \left\lfloor \frac{(k-1)n + \sqrt{(k-1)^2n^2 + 4(K_m^2 - (K-1)n}}{2(K_m^2 - (k-1)n)} \right\rfloor$$

$$l \leftarrow mK_m - 1$$

Paso 1: Encontrar un polinomio $Q(x, y) \in \mathbb{F}_q[x, y]$ tal que $deg(1, v)(Q(x, y)) \leq l$, es decir, encontrar los valores de sus coeficientes $a_{i,j}$ con $i + vj \leq l$.

Paso 2: Encontrar y retornar todos los polinomios $p(x) \in \mathbb{F}_q[x]$ de grado $\leq k - 1$ tales que $(y - p(x))|Q(x, y)$. Devolver aquellos polinomios que cumplan que $p(x_i) = y_i$ por lo menos en K_m valores de $i \in [n]$.

2.4. La interpolación empleando el algoritmo de Koetter

El objetivo del proceso de interpolación como ya sabemos es determinar el polinomio $Q(x,y) \in \mathbb{F}_q[x,y]$, es decir determinar el polinomio que pasa por los puntos (x_i, y_i) con multiplicidad m, de mínimo grado ponderado $(1,v)$. El polinomio debe estar expresado como la suma de los primeros $C+1$ monomios según un orden basado en el grado ponderado de los mismos. Es decir, el orden de los monomios que eligamos ha de cumplir

$$\deg_{(1,v)}(x^{i1}, y^{j1}) < \deg_{(1,v)}(x^{i2}, y^{j2}) \Rightarrow \text{rango}(x^{i1}, y^{j1}) < \text{rango}(x^{i2}, y^{j2}) \tag{31}$$

Puesto que, en general habrá más de un polinomio que cumpla la condición de interpolación es interesante retornar aquel que tenga menor grado para de esta forma simplificar el proceso posterior de factorización.

El hecho de que $Q(x,y)$ se exprese como la suma de $C+1$ monomios no implica necesariamente que tenga el grado del monomio de mayor grado, puesto que eventualmente, el polinomio que cumpla las condiciones interpolación puede tener los coeficientes de mayor grado iguales a 0, debido a que al cumplir una restricción pueden estar cumpliéndose simultáneamente más de una.

Por ejemplo si estamos buscando un polinomio $Q(x,y) \in \mathbb{F}_q[x,y]$ de mínimo (1,1)-grado ponderado que pase por los puntos (α,α) y $(\beta,\beta) \in \mathbb{F}_q$ con multiplicidad 1. Por tanto tenemos un total de $2\binom{1+1}{2} = 2$ restricciones y necesitaríamos un máximo de 3 monomios para realizar el proceso de interpolación. El polinomio que cumple estas condiciones es $Q(x,y) = x - y$, no obstante, este polinomio también cumpliría el problema de interpolación, cualquier punto de la forma $(\gamma,\gamma) \in \mathbb{F}$ sin necesidad de utilizar más coeficientes.

En este apartado vamos a describir el algoritmo de Koetter, el cual proporciona una solución al problema de la interpolación. Según lo desarrollado hasta el momento una primera aproximación para resolver el problema de la interpolación sería plantear un sistema de C ecuaciones lineales con $C+1$ incógnitas que serán los coeficientes de $Q(x,y)$. La propia naturaleza del problema sugiere un algoritmo de interpolación sencillo (algoritmo 2) el cual realiza C iteraciones.

En el primer paso, el algoritmo inicializa cada uno de los polinomios candidatos Q_j, siendo cada uno de ellos la variable y a la potencia i, para $0 \leq i \leq dy$. Donde dy es un parámetro de entrada que identifica la máxima potencia de la variable y de los monomios que utilizaremos en el proceso de interpolación. Dicha potencia máxima se designa como hemos dicho por el parámetro dy y será aquel valor que cumpla:

$$\text{rango}(y^{dy}) \leq C \leq \text{rango}(y^{dy+1}) \tag{32}$$

según el $(1,v)$-orden ponderado elegido.

Este paso garantiza que el grado $(1,v)$-ponderado de ninguna combinación lineal de estos polinomios será mayor que el necesario, por lo que respecta a la variable y. El resto del algoritmo se encarga de aplicar las restricciones lineales:

$$D_{r,s}Q(x_i, y_i) = 0 \tag{33}$$

o lo que es lo mismo

$$Q_{r,s}^{(x_i,y_i)} = \sum_k \sum_j \binom{k}{r}\binom{j}{s} a_{k,j} x_i^{k-r} y_i^{j-s} = 0 \tag{34}$$

$\forall\ i \in [n]$, $\forall\ r,s \geq 0 : r+s < m$, en cada uno de los $dy+1$ polinomios candidatos. Al finalizar cada iteración externa, para un punto (x_i, y_i), todos los candidatos contendrán una singularidad de orden m en ese punto, es decir, los polinomios candidatos pasarán por ese punto con multiplicidad m.

En los primeros pasos se dedica a buscar el candidato de menor $(1,v)$-grado ponderado que no cumpla la restricción exigida, $q_{r,s}^{(x_i,y_i)} = 0$ (para unos valores r,s dados), el cual podemos denotar como Q_{lmin}. Seguidamente se fuerza a cumplir esa restricción $q_{r,s}^{(x_i,y_i)} = 0$ a todos los polinomios que no la cumplían, excepto al que hemos identificado como el de menor grado ponderado que no cumplía la condición, el cual se usa como elemento pivote.

Si observamos la ecuación de actualización,

$$Q'_j = \Delta_{lmin} * Q_j - \Delta_j * Q_{lmin} \tag{35}$$

donde $\Delta_j = D_{r,s}Q(x_i, y_i)$, viendo que, después de la actualización el valor de la condición sí que es cero como se buscaba

$$D_{r,s}Q'_j(x_i,y_i) = (D_{r,s}Q_{lmin}(x_i,y_i))*(D_{r,s}Q_j(x_i,y_i)) - (D_{r,s}Q_j(x_i,y_i))*(D_{r,s}Q_{l_{min}}(x_i,y_i)) = 0 \tag{36}$$

Este paso ni incrementa el $(1,v)$-grado ponderado de Q_j (puesto que $\deg^{(1,v)}Q_{lmin} < \deg^{(1,v)}Q_j$) ni se modifican las imposiciones de restricciones anteriores (ya que al finalizar cada iteración del bucle todos los polinomios las cumplían y una combinación lineal de éstos también las cumplirán). Por último el algoritmo fuerza la condición correspondiente a la restricción $D_{r,s}Q_{lmin}(x_i,y_i) = 0$. Puesto que es el polinomio de menor $(1,v)$-grado ponderado, la única manera de realizar esto es multiplicar por alguno de los factores $(x-x_i)$ ó $(y-y_i)$. Pero multiplicar por este último factor implicaría incrementar el $(1,v)$-grado ponderado en v unidades, mientras que el primer factor solo lo hará en una unidad. Además, el valor de la máxima potencia de y ya es parámetro de entrada y usado al principio del algoritmo. Este proceso se repetirá para todos los valores de r,s tales que su suma sea menor que m, teniendo al final del bucle que todos los polinomios cumplen las restricciones asociadas al punto (x_i,y_i). El proceso se repetirá para todos los puntos, obteniendo como resultado que todos los polinomios utilizados pasan por los n puntos con la multiplicidad requerida.

El algoritmo tiene que retornar el polinomio de mínimo $(1,v)$-grado ponderado de todos los candidatos Q_j. Hay que tener en cuenta que, pese a que cualquiera de los polinomios Q_l cumplen las restricciones lineales el algoritmo ha de retornar un polinomio Q_{lmin} tal que

$$Q_{lmin} \in \mathbb{F}_q[x,y]\ \backslash \mathbb{F}_q[x]^4 \tag{37}$$

[4] $l \in A \backslash B$ si y sólo si $l \in A$ y $l \notin B$

de lo contrario, no sería un polinomio válido a entregar en el proceso de factorización. Una observación importante es que esto no sucederá si $dy < 1$, por lo tanto es otra condición a tener en cuenta a la hora de determinar el coste de interpolación C, derivado de las multiplicidades de los puntos de interpolación, es decir, debe cumplirse que

$$C(m) > \text{rango}(y) \tag{38}$$

Ya que sabemos que debemos elegir un orden ponderado de monomios, la condición anterior sugiere que, de los ordenes ponderados presentados en la sección 2.2.1, resulta recomendable seleccionar aquel de mínimo rango del monomio y. Ya que

$$\text{rango}(y^j)|_{\text{revlex}} = \text{rango}(y^j)|_{\text{lex}} + j \tag{39}$$

está claro que el orden de $M[x, y]$ más recomendable es el orden lexicográfico. En este caso, el valor de dy a partir de la ecuación 32 y a partir de rango(y^{dy}) según el orden lexicográfico ponderado será

$$dy = \left\lfloor \frac{1 + \sqrt{1 + \frac{8C}{k-1}}}{2} \right\rfloor - 1 \tag{40}$$

Una expresión más detallada para la máxima potencia en y puede encontrarse en [11].

Algoritmo 2 Algoritmo de Koetter

Entrada: Conjunto de puntos (x_i, y_i); $i = 1, \cdots, n$; orden de interpolación; m; orden monomios $(1, v)$; dy **Salida:** Polinomio que satisface el problema $Q(x, y)$

Inicialización: $Q_j = y^j$ para $j = 0$ hasta dy

Para $i = 1$ hasta n

Para $(r, s) = (0, 0)$ hasta $m_i - 1, 1)$

para $j = 0$ hasta dy

$\Delta_j = \text{coeff}(Q_j(x + x_i, y + y_i), x^r\ y^s)$

$j^* = argmin\{Q_j : j \in J\}$

$f = Q_j$

$\Delta = \Delta_{j^*}$

si $(j \neq j^*)$

$Q_j = \Delta Q_j - \Delta_j f$

si no $(j = j^*)$

$Q_j = (x - x_i)f$

$Q(x, y) = min_j\{Q_j(x, y)\}$

2.5. El algoritmo de factorización de Roth-Ruckenstein

La segunda fase del algoritmo como hemos comentado consiste en factorizar el polinomio $Q(x, y)$ retornado en el proceso de interpolación e identificar todos los factores de la

forma $y - p(x)$. A los polinomios $p(x)$ se le denomina candidatos. El resultado final serán aquellos de los candidatos que cumplan que $\deg(p(x)) < k$ e $y_i = p(x_i)$ por lo menos en K_m valores, o desde el punto de vista de la decodificación de un código Reed-Solomon $RS(n, k)$, aquel conjunto de palabras código asociadas a los candidatos $(f(x_1), f(x_2), \cdots, f(x_n))$, que estén a distancia $t_m = n - K_m$ de la palabra recibida, $(y_1, y_2, \cdots, y_n)$.

Una primera aproximación consistiría en factorizar completamente el polinomio $Q(x, y)$ en factores irreducibles de $\mathbb{F}_q[x, y]$, pero dado que solo estamos interesados en conocer factores de la forma $y - p(x)$ favorece que no sea necesario realizar esta operación. Una observación clave es el hecho de que determinar dichos factores equivale a factorizar el polinomio $Q(x, y) \in \mathbb{F}_q[x, y]$ en $\mathbb{F}_q[x][y]$, que es un problema mucho más sencillo, de tal manera que en este paso se irá evaluando el polinomio bivariable en 0 en x o y de manera que se trabaja como si fuese un polinomio univariable en y o x respectivamente.

En este apartado se va a presentar uno de los algoritmos más eficientes que se conoce en la actualidad para resolver el problema de la factorización, llamado algoritmo de Roth-Ruckenstein. Este algoritmo determina los coeficientes de los polinomios $p(x)$ de forma iterativa, determinando las raíces de polinomios univariables en $\mathbb{F}_q$.

Para ello necesitamos definir la aplicación $\langle\langle\rangle\rangle$. La notación $\langle\langle Q(x, y)\rangle\rangle$ indica el coeficiente de la mayor potencia de x divide a $Q(x, y)$. Esto es, si $x^m | Q(x, y)$ pero $x^{m+1} \nmid Q(x, y)$, entonces

$$\langle\langle Q(x, y)\rangle\rangle = \frac{Q(x, y)}{x^m} \tag{41}$$

Por lo tanto,

$$Q(x, y) = \langle\langle Q(x, y)\rangle\rangle \, x^m \tag{42}$$

para algún $m \geq 0$. Aunque $Q(0, y)$ puede ser igual a cero, sin embargo, $\langle\langle Q(0, y)\rangle\rangle$ es un polinomio distinto de cero en y (por ejemplo $Q(x, y) = xy,\ Q(0, y) = 0$ pero $\langle\langle Q(0, y)\rangle\rangle = y$).

Suponemos

$$p(x) = a_0 + a_1 x + \cdots + a_v x^v \tag{43}$$

es una raíz y de $Q(x, y)$. Veremos como los coeficientes $a_0, a_1, \cdots, a_v$ pueden ser extraídos de forma iterativa, obteniendo un coeficiente en cada paso. Con el siguiente lema se muestra como se determina el valor del coeficiente a_0.

Lema 2.4 *Si $y - p(x) | Q(x, y)$, entonces $y = p(0) = a_0$ es una raíz de la ecuación $Q_0(0, y) = 0$, donde $Q_0(x, y) = \langle\langle Q(x, y)\rangle\rangle$*

Para este lema, el conjunto de posibles valores de los coeficientes a_0 de (43) son las raices del polinomio $Q_0(0, y)$. El algoritmo ahora trabaja por inducción "pelando" capas, dejando una estructura en la cual a_1 puede ser encontrada de forma similar, entonces a_2, y así sucesivamente. En ello se basa el siguiente teorema el cual define el proceso de inducción y extiende el 2.4.

Teorema 2.5 *Dado* $Q_0(x,y) = \langle\langle Q(x,y)\rangle\rangle$, *y* $p_0 = p(x) = a_0 + a_1 x + \cdots + a_v x^v \ \in \mathbb{F}_v[x]$. *Para* $j \geq 1$ *se define*

$$p_j(x) = \frac{(p_{j-1}(x) - p_{j-1}(0))}{x} = a_j + \cdots + a_v x^{v-j} \tag{44}$$

$$T_j(x,y) = Q_{j-1}(x, xy + a_{j-1}) \tag{45}$$

$$Q_j(x,y) = \langle\langle T_j(x,y)\rangle\rangle \tag{46}$$

Entonces para cualquier $j \geq 1$, $(y - p(x))|Q(x,y)$ *si y solo si* $(y - p_j(x))|Q_j(x,y)$.

A los teoremas anteriores le sigue el siguiente corolario. Si $(y - f(x))|Q(x,y)$, entonces el coeficiente a_i es una solución de la ecuación $Q_i(0,y) = 0$ para $i = 1, 2, \cdots, v$. Ya que $p_j(0) = a_j$, este corolario indica que el coeficiente a_j se puede encontrar mediante la búsqueda de las raíces de la ecuación $Q_j(0,y) = 0$

En general, el algoritmo 3 navega a través de una estructura de árbol determinando los coeficientes de los posibles factores lineales de $Q(x,y)$, como podemos observar en el ejemplo de la figura 3. En el pseudocódigo como entradas tenemos el polinomio bivariable $Q(x,y)$ hallado en el proceso de interpolación y un entero positivo D, y devuelve como salida una lista de todas las posibles y-raices de $Q(x,y)$ de grado $\leq D$.

Algoritmo 3 Algoritmo de Roth-Ruckenstein

Entrada: Polinomio bivariable $Q(x,y) \in \mathbb{F}_q[x,y]$, valor entero D (máximo grado de $p(x)$)
Salida: Lista de todos polinomios $p(x)$ de grado $\leq D$ tal que $(y-p(x))|Q(x,y)$

Inicialización: $p(x)=0,\ u=deg(p)=-1$
Lista donde serán guardados los polinomios
Llamada a arbolrothruck$(Q(x,y),u,p)$

Funcion arbolrothruck
Entrada: $Q(x,y)$, $p(x)$ y u (grado de p)
Salida: Lista de polinomios
si $(Q(x,0)=0)$ entonces
añadimos $p(x)$ a la lista de salida
si no
si $(u<D)$
Encontrar todas las raíces en $\mathbb{F}_q$ del polinomio univariable $Q(0,y)$.
Para cada raíz α de $Q(0,y)$ hacer
$Q_{\text{nuevo}}(x,y)=Q(x,xy+\alpha)$
Calcular $\langle\langle Q_{new}(x,y)\rangle\rangle$
Añadir el nuevo coeficiente al polinomio, $p_{u+1}=\alpha$
Llamar a arbolrothruck $(\langle\langle Q_{new}(x,y)\rangle\rangle, u+1, p)$, creando para cada α una nueva rama del árbol
si no
Rama del árbol finalizada sin ninguna salida.

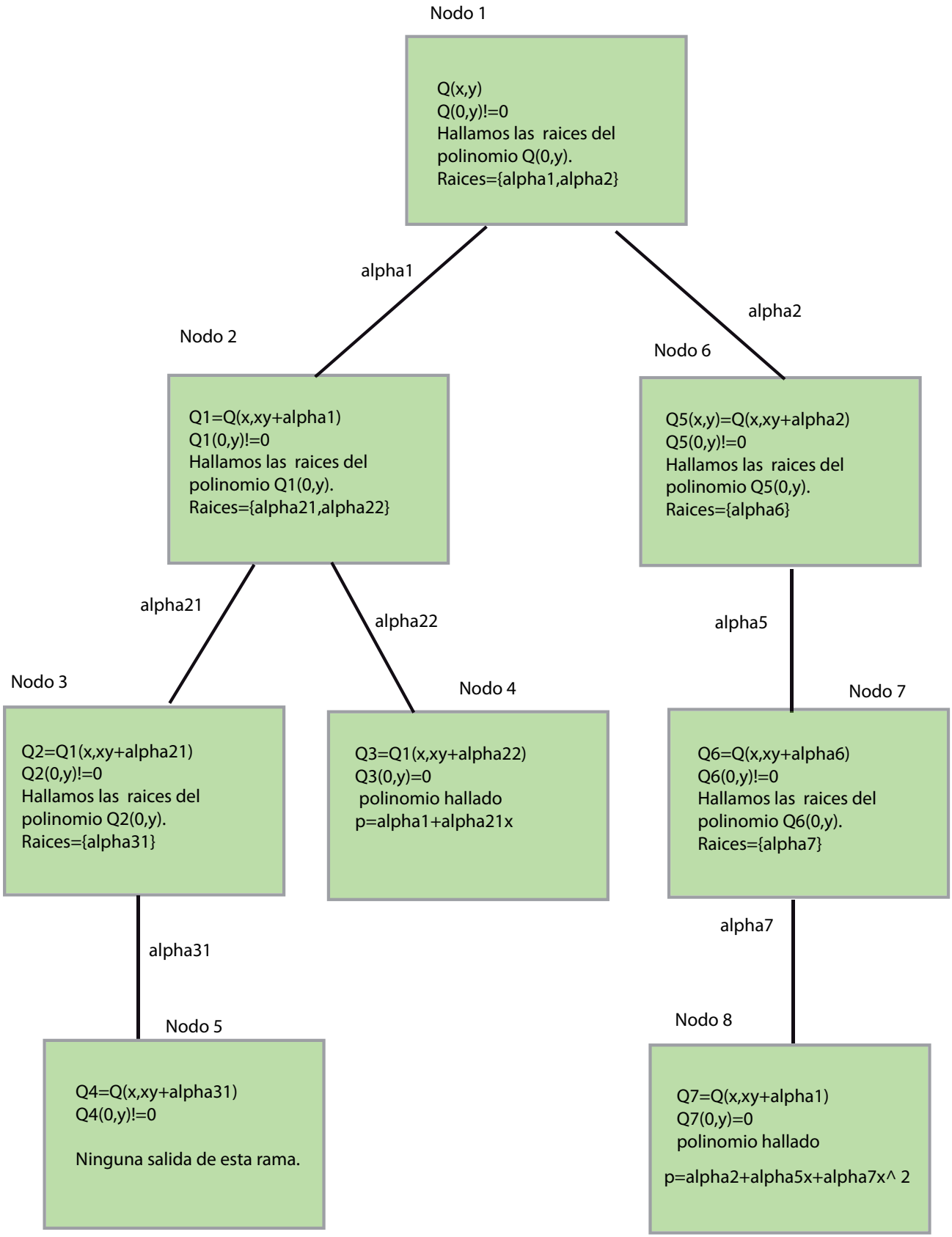


Figura 3: Ejemplo del algoritmo Roth-Ruckenstein con D=2

2.6. Decodificación *soft* para códigos Red-Solomon

Hasta este punto del proyecto se ha hablado de decodificación *hard*, haciendo uso de la estructura algebraica del código empleando símbolos que pueden ser interpretados como elementos en un cuerpo finito o de Galois. En este apartado se propone hacer uso de la información del canal para llevar a cabo la decodificación *soft* de los códigos Reed-Solomon. La idea propuesta consiste en transformar la información aportada por el canal en un conjunto de restricciones lineales, esto se traduce en determinar los valores de la multiplicidad m_i para cada uno de los puntos a interpolar $(x_i, y_i) \in \mathbb{F}_q$, al contrario de lo que sucede con el algoritmo original de Guruswami-Sudan, en los que todos los puntos tienen la misma multiplicidad m. Este problema resuelto por una extensión del algoritmo de Guruswami-Sudan, es llamado algoritmo de Koetter-Vardy(KV). En las siguientes figuras, observamos como la segunda etapa de los algoritmos de Guruswami-Sudan y Koetter-Vardy son prácticamente idénticas. No obstante, existe una ligera diferencia en la definición de los requisitos exigidos al polinomio $Q(x, y)$, puesto que en el caso de KV los valores de las multiplicidades para los puntos de interpolación no tienen porqué tener el mismo valor.

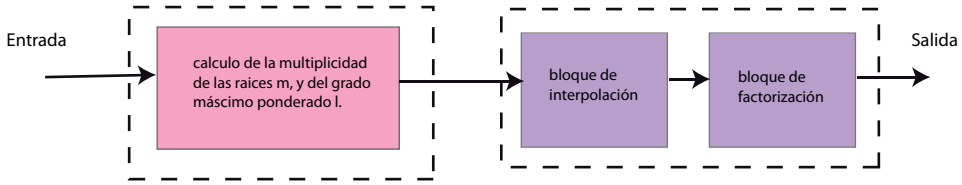


Figura 4: Diagrama bloques de Guruswami-Sudan

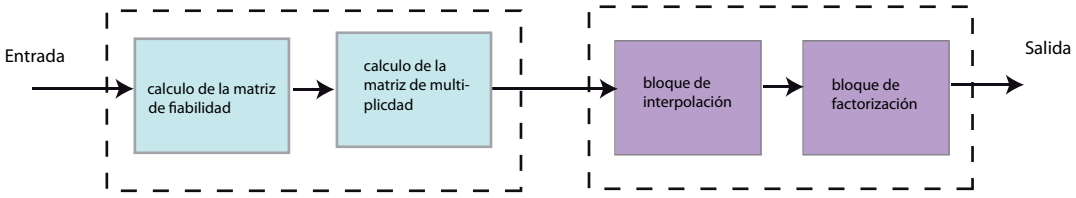


Figura 5: Diagrama bloques de Koetter-Vardy

En el algoritmo KV, como observamos en la figura 5, se utiliza un asignación de probabilidades usando información del canal, para posteriormente utilizarlas para la asignación de las multiplicidades a cada punto de interpolación. Después de esto se utiliza la interpolación y factorización convencional vistas hasta este momento.

Un canal sin memoria puede ser modelado por una terna compuesta por un alfabeto de entrada χ, un alfabeto de salida υ y un conjunto de $|\chi|$ funciones $f(.|x) : \upsilon \to \mathbb{R}$. La entrada y la salida del canal son convencionalmente variables aleatorias X e Y, respectivamente. Si Y es continuo entonces $f(.|x)$ es una función densidad de probabilidad continua. En cambio si Y es discreto, entonces es una función de distribución discreta.

En cualquier caso, el decodificador puede calcular la probabilidad de que $\alpha_i \in \chi$ se haya transmitido dado que observa a la salida $\beta_j \in \upsilon$. Suponiendo que son variables

uniformemente distribuidas y usando el teorema de Bayes

$$P(X = \alpha_i | Y = \beta_j) = \frac{f(\beta_j|\alpha_i)P(X = \alpha_i)}{\sum_{x\in\chi} f(\beta_j|x)P(X = x)} = \frac{f(\beta_j|\alpha_i)}{\sum_{x\in\chi} f(\beta_j|x)} \tag{47}$$

donde la igualdad se cumple si suponemos que X está uniformemente distribuida sobre χ.

Para un código Reed-Solomon $RS(n, k)$, en el que el alfabeto de entrada es el cuerpo sobre el cual se producen los símbolos, $\chi = \mathbb{F}_q$ y consideramos $\alpha_1, \alpha_2, \cdots, \alpha_q$ como un orden de sus q elementos. Si se recibe la palabra $\beta = (\beta_1, \beta_2, \cdots, \beta_n)$ usando la fórmula (47) podemos calcular los valores

$$\pi_{i,j} = P(\chi = \alpha_i | \upsilon = \beta_i) \qquad i = 1, 2, \cdots, q, \qquad j = 1, 2, \cdots, n \tag{48}$$

y disponerlos en una matriz de dimensión $q\times n$, llamada matriz de fiabilidad, designada por $\mathbf{\Pi}$ con elementos $\pi_{i,j}$. Es conveniente usar la notación $\mathbf{\Pi}(\alpha, j)$ para referirnos al elemento en la fila indexada por α y la columna indexada por j. Se supone que la matriz de fiabilidad está dada siempre, de alguna manera, como entrada al algoritmo de decodificación.

En el caso de utilizar la información del canal para realizar una decodificación *hard*, la palabra que se pasará al decodificador del código de Reed-Solomon será

$$(\arg\max_{\alpha_i}(\pi_{i,1}), \cdots, \arg\max_{\alpha_i}(\pi_{i,n})) \tag{49}$$

En el caso de realizar una decodificación con información de canal se utilizará toda la matriz $\mathbf{\Pi}$ para determinar las multiplicidades de los puntos de interpolación. Una segunda matriz es empleada, una matriz de multiplicidad, $\mathbf{M}$ asociada a la de fiabilidad $\mathbf{\Pi}$, de dimensiones $q \times n$ la cual es una matriz donde $m_{i,j}$ representa la multiplicidad de interpolación asociada al punto (α_i, β_j). El paso clave del algoritmo que se describe a continuación es proporcionar una asignación de la matriz de fiabilidad $\mathbf{\Pi}$ a la matriz de multiplicidad $\mathbf{M}$.

Recordemos que el punto principal del teorema de interpolación es, que debe haber más grados de libertad(variables) que limitaciones. El número de restricciones introducidas por una multiplicidad $m_{i,j}$ como ya hemos visto es igual a $\binom{m_{i,j}+1}{2}$.El número total de restricciones asociadas a una matriz de multiplicidad $\mathbf{M}$ es llamada el coste de $\mathbf{M}$, denotado $C(M)$, donde

$$C(M) = \frac{1}{2}\sum_{i=1}^{q}\sum_{j=1}^{n} m_{i,j}(m_{i,j} + 1) \tag{50}$$

Como antes habíamos definido, $C(v, l)$ el numero de monomios de $(1, v)$-grado ponderado menor o igual a l. Entonces por el teorema de interpolación existe una solución de interpolación si

$$C(v, l) > C(M) \tag{51}$$

Será conveniente representar los vectores en $\mathbb{F}_q^n$ como un indicador de matrices sobre los reales. Dada la palabra $\mathbf{v} = (v_1, v_2, \cdots, v_n) \in \mathbb{F}_q^n$ ésta puede expresarse como una matriz

$q \times n$, denotada por $[\mathbf{v}] = ([v]_{i,j})$ de la siguiente forma

$$[v]_{i,j} = \begin{cases} 1 & \text{si } v_j = \alpha_i \\ 0 & \text{en caso contrario} \end{cases} \tag{52}$$

Sea $\mathbf{A} = (a_{i,j})$ y $\mathbf{B} = (b_{i,j})$ dos matrices de dimensión $q \times n$ definidas sobre el mismo cuerpo. El producto interno de $\mathbf{A}$ y $\mathbf{B}$,$\langle \mathbf{A}, \mathbf{B} \rangle$ se define como

$$\langle \mathbf{A}, \mathbf{B} \rangle = \text{traza}(\mathbf{A}, \mathbf{B}) = \sum_{i=1}^{q} \sum_{j=1}^{n} a_{i,j} b_{i,j} \tag{53}$$

Ahora usamos esto, para definir el coste de un vector $\mathbf{v} = (v_1, v_2, \cdots, v_n)$ con respecto a la matriz de multiplicidad M,

$$S_M = (\mathbf{v}) = \langle M, [\mathbf{v}] \rangle \tag{54}$$

El coste representa así la multiplicidad total de todos los puntos relacionados con el vector $\mathbf{v}$. Dado un coste C, definimos el conjunto $\mathcal{M}(C)$ como el conjunto de matrices con elementos no negativos cuyo coste es igual a C:

$$\mathcal{M}(C) = \{M \in \mathbb{Z}^{qxn} : m_{i,j} \geq 0 \text{ y } \frac{1}{2} \sum_{i=1}^{q} \sum_{j=1}^{n} m_{i,j}(m_{i,j} + 1) = C\} \tag{55}$$

El problema que ocuparía ahora es seleccionar una matriz de multiplicidad $\mathbf{M}$ que maximice el coste $S_M(c)$ para una palabra código trasmitida $\mathbf{c}$.

El algoritmo 4 realiza esta operación, determinando la mejor matriz de multiplicidad $\mathbf{M}(\mathbf{\Pi}, s)$ para un conjunto de puntos de interpolación tales que la suma de sus multiplicidades es igual a s,

$$s \triangleq \sum_{i=1}^{n} m_i \tag{56}$$

Dada una palabra $u \in RS(n,k)$ y una matriz de fiabilidad, $\mathbf{\Pi}$, el algoritmo 4 retorna la matriz $\mathbf{M}(\mathbf{\Pi}, s)$ que maximiza $\langle M, [\mathbf{u}] \rangle$, de entre todas las matrices de elementos no negativos de dimensión $q \times n$ con el mismo coste. Es decir, si C es el coste de $\mathbf{M}(\mathbf{\Pi}, s)$,

$$M(\mathbf{\Pi}, s) = \arg \max_{M \in \mathcal{M}(C)} \langle M, [\mathbf{v}] \rangle \tag{57}$$

Algoritmo 4 Cálculo de la matriz de multiplicidad

Entrada: Matriz de fiabilidad $\mathbf{\Pi}$ de dimensión $q \times n$ y un entero $s > 0$ indicando el número total de puntos de interpolación (contados con sus multiplicidades).

Salida: Matriz de multiplicidad $\mathbf{M}(\mathbf{\Pi}, s)$

Inicialización:

$$\mathbf{\Pi}^* \leftarrow \mathbf{\Pi}, \qquad M \leftarrow \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix} \tag{58}$$

Hacer:

Encontrar la posición (i, j) del máximo $\pi^*_{i,j}$ en $\mathbf{\Pi}^*$ y asignar

$$\begin{aligned} \pi^*_{i,j} &\leftarrow \frac{\pi_{i,j}}{m_{i,j}+2} \\ m_{i,j} &\leftarrow m_{i,j}+1 \\ s &\leftarrow s-1 \end{aligned}$$

Mientras $s > 0$

Algoritmo 5 Cálculo simplificado de la matriz de multiplicidad

Entrada: Matriz de fiabilidad $\mathbf{\Pi}$ de dimensión $q \times n$ y un valor real $\lambda < 0$.
Salida: Matriz de multiplicidad $\mathbf{M}(\mathbf{\Pi}, s)$

para $1 \leq i \leq q$ hacer
  para $1 \leq j \leq n$ hacer

$$m_{i,j} \leftarrow \lfloor \lambda \pi_{i,j} \rfloor \tag{59}$$

  fin para
fin para

Para $s \to \infty$ la matriz de multiplicidad generada por el algoritmo 4, $\mathbf{M}(\mathbf{\Pi}, s) = (m_{i,j})$, se vuelve proporcional a la matriz de fiabilidad, $\mathbf{\Pi} = (\pi_{i,j})$. Dicho de otra forma, para cualquier número real $\lambda > 0$ existe un s tal que los elementos de la matriz $\mathbf{M}(\mathbf{\Pi}, s)$ cumplen que

$$m_{i,j} = \lfloor \lambda \pi_{i,j} \rfloor \tag{60}$$

El teorema anterior sugiere una manera de generar la matriz de multiplicidad (algoritmo 5), simplemente aplicando un factor de escala a la matriz de fiabilidad, cosa que parece intuitiva. No obstante, este método no es exactamente equivalente al algoritmo 4 para matrices de fiabilidad que tengan una precisión finita en sus entradas no existe ningún valor de λ para el que se cumpla (60).

Una vez presentado el método para transformar la información del canal a multiplicidades, el algoritmo de interpolación recibirá como datos de entrada los puntos $(x_i, y_i) = (\alpha_i, \beta_j)$ con multiplicidades $m_i = m_{i,j}$, para $i \in [n]$, $n = |\{m_{i,j} : m_{i,j} > 0\}|$.

3. Implementaciones

3.1. Implementaciones secuenciales

En esta primera versión secuencial se ha desarrollado todo el software necesario para implementar la decodificación de lista. Se han implementado los principales algoritmos

desarrollados en la memoría así como diferentes bibliotecas que dan soporte a la resolución de estos algoritmos.

En los conceptos teóricos se ha visto la importancia tanto de los polinomios bivariables como de los cuerpos Finitos o de Galois. Por este motivo han hecho falta principalmente diseñar dos bibliotecas en C, una de ellas para el tratamiento de polinomios bivariables y otra para el de Cuerpos Finitos. Ambas bibliotecas se apoyan en unas antiguas desarrolladas en mi PFC [24] tratándose de unas implementaciones mejoradas.

3.2. JNI

En última instancia la finalidad práctica del proyecto desarrollado (implementado en C) es realizar la decodificación de forma eficiente en tiempo de ejecución en el código de *fingerprinting*(implementado en Java). Para tal finalidad se ha tenido que desarrollar una interfaz entre Java y C para poder incorporar el decodificdor implementado al software proporcionado.

Para este propósito se ha hecho uso de *Java Native Interface* (JNI) el cual es un marco de programación que permite que un programa escrito en Java pueda interactuar con programas escritos en otros lenguajes como C, C++ y ensamblador. JNI es parte de la máquina virtual Java y permite invocaciones en ambos sentidos: aplicaciones Java pueden invocar código nativo escrito en otro lenguaje y viceversa. En el proyecto el uso de JNI es como nexo entre Java y un lenguaje nativo de la máquina en este caso C.

3.3. Programación paralela y OpenMP

Tradicionalmente se han implementado programas secuenciales los cuales funcionan en dispositivos con un único núcleo o en un único hilo, de este modo se dispone de un problema compuesto por un conjunto de instrucciones las cuales se ejecutan una detrás de otra en el núcleo del dispositivo.

Sin embargo hoy en día la mayoría de procesadores actuales y dispositvos son multinúcleo. Por tanto se puede decir que el paralelismo es el futuro y presente de la computación. En el cálculo en paralelo se puede realizar la ejecución de más de una instrucción al mismo tiempo usando más de un procesador o núcleo del dispositivo. Con ello se pretende reducir el tiempo total de ejecución distribuyendo la carga de trabajo entre los procesadores disponibles, obteniendo un alto rendimiento o mayor velocidad al ejecutar un programa. No solo esa es la razón de ser de la programación paralela, la construcción de aplicaciones más complejas han requerido computadoras más rápidas. Existen distintos paradigmas de paralelización; podemos tener paralelización a nivel de bit, de instucción, de datos y de tareas.

En el presente trabajo se han utilizado directivas de compilación OpenMP para la programación multiproceso expresando paralelismo multihilo en sistemas de memoria compartida. En este modelo de memoria compartida los procesadores interactúan entre ellos mediante el uso de variables compartidas y mecanismos de sincronización.

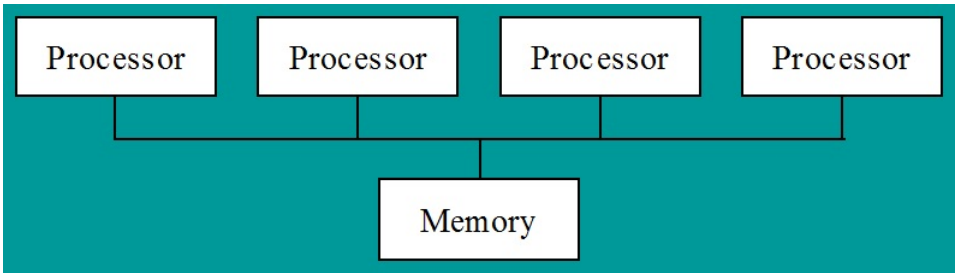


Figura 6: Sisttema de memoria compartida

Inicialmente solamente está activo el hilo maestro el cual ejecuta el código secuencial, para ejecutar el código paralelo este hilo maestro crea hilos adicionales de tal modo que al final del código paralelo mueren o se suspenden. En nuestro caso, tras un exhaustivo análisis algorítmico, se ha identificado las partes del algoritmo más susceptibles de ser paralelizadas eficientemente. Se han escogido las herramientas de OpenMP que han permitido conseguir el mejor rendimiento posible a la paralelización propuesta de cierto bucle del código. Para la paralelización de un bucle for hay que tener en cuenta una serie de consideraciones: no debe haber dependencia entre las distintas iteraciones del bucle, quedan prohibidas las instrucciones break, exit(),goto,... y el bucle debe estar escrito en forma canónica.

De este modo utilizando la directiva:

```
#pragma omp parallel for
 for (i = 0; i < N; i++)
    a[i] = b[i] + c[i];
```

el hilo maestro crea hilos adicionales para cubrir las iteraciones del bucle. Cada hilo tiene su propio contexto de ejecución y por defecto, todas las variables son compartidas, salvo el iterador del bucle. Podemos especificar qué variables serán privadas a cada hilo y cuales compartidas utilizando a continuación del pragma

```
private (j)
```

en cuyo caso tanto el iterador del bucle como la variable j indicada serán variables privadas y después del bucle el valor de estos parámetros es indefinido ya que no se conoce el valor a la salida de las variables privadas.

Las iteraciones de los bucles pueden distribuirse estática o dinámicamente entre los hilos; en nuestro caso, las pruebas realizadas con ambas posibilidades no han dado un claro mejor candidato debido a la variabilidad del rendimiento en función de la instancia de problema, por lo que hemos adoptado por defecto la solución estática.

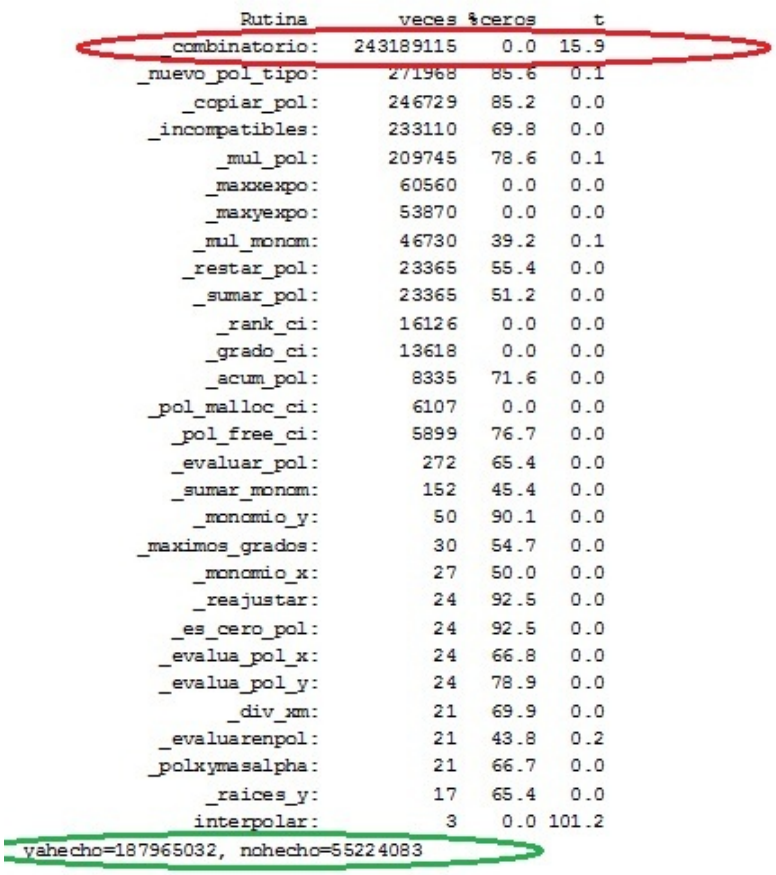
4. Resultados

En el presente capítulo se presentarán y analizarán los resultados obtenidos tras la implementación del algoritmo. En primer lugar se desarrolló una primera versión secuencial del código, la cual fue estudiada y optimizada en diferentes aspectos. Tras esta primera implementación y validación del software desarrollado el primer paso consistió en perfeccionar el código con el fin de reducir el tiempo de ejecución. En la tabla 3 podemos observar la mejora conseguida tras la optimización del código, en esta tabla se han representado el tiempo que de ejecución de cada uno de los dos códigos en cuatro problemas distintos de detección de usuarios confabuladores y el speedup obtenido para cada uno de los ejemplos.

	Tiempo secuencial sin optimizar (s)	Tiempo secuencial optimizado (s)	Speedup
Ejemplo 1	2,68	0,375	7,14
Ejemplo 2	1,55	0,183	8,46
Ejemplo 3	8,77	0,76	11,5
Ejemplo 4	44,67	19,8	2,25

Tabla 3: Comparativa entre el código secuencial optimizado y sin optimizar

Para llegar a esa mejora, se optimizaron distintas funciones implementadas pero la principal influencia en el resultado fue la mejora de una función la cual se encarga de calcular el combinatorio de dos números. Ello es debido a que esta función es una de las que más número de veces es llamada durante la ejecución del software tal y como podemos observar en la figura 7 resaltado en rojo



Rutina	veces	%ceros	t
_combinatorio:	243189115	0.0	15.9
_nuevo_pol_tipo:	271968	85.6	0.1
_copiar_pol:	246729	85.2	0.0
_incompatibles:	233110	69.8	0.0
_mul_pol:	209745	78.6	0.1
_maxxexpo:	60560	0.0	0.0
_maxyexpo:	53870	0.0	0.0
_mul_monom:	46730	39.2	0.1
_restar_pol:	23365	55.4	0.0
_sumar_pol:	23365	51.2	0.0
_rank_ci:	16126	0.0	0.0
_grado_ci:	13618	0.0	0.0
_acum_pol:	8335	71.6	0.0
_pol_malloc_ci:	6107	0.0	0.0
_pol_free_ci:	5899	76.7	0.0
_evaluar_pol:	272	65.4	0.0
_sumar_monom:	152	45.4	0.0
_monomio_y:	50	90.1	0.0
_maximos_grados:	30	54.7	0.0
_monomio_x:	27	50.0	0.0
_reajustar:	24	92.5	0.0
_es_cero_pol:	24	92.5	0.0
_evalua_pol_x:	24	66.8	0.0
_evalua_pol_y:	24	78.9	0.0
_div_xm:	21	69.9	0.0
_evaluarenpol:	21	43.8	0.2
_polxymasalpha:	21	66.7	0.0
_raices_y:	17	65.4	0.0
interpolar:	3	0.0	101.2

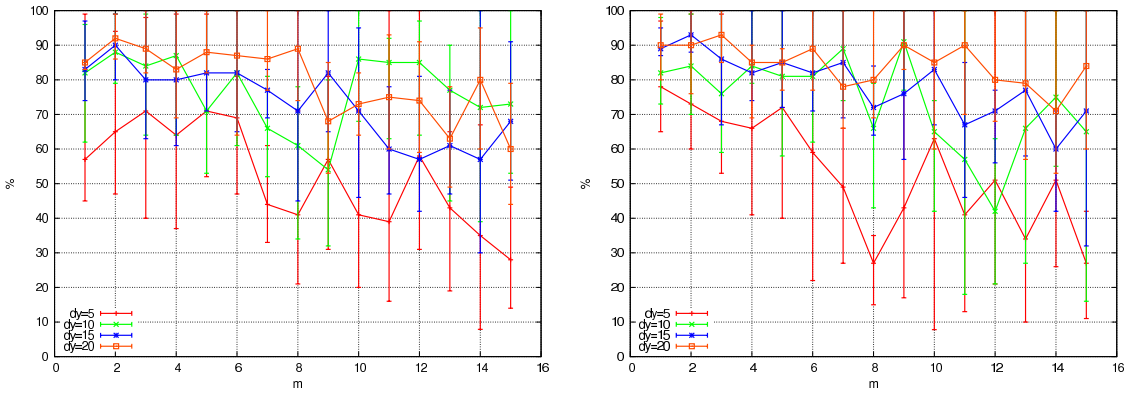
yahecho=187965032, nohecho=55224083

Figura 7: Tiempo de ejecución de diferentes funciones

De este modo se implementó un método de cálculo de números combinatorios más eficiente que el ya implementado. Otro dato interesante que se puede extraer de los resultados es la gran cantidad de veces que se calcula un combinatorio el cual ya se ha calculado anteriormente(marcado en verde en la figura 7). De este modo se creó un vector para almacenar combinatorios ya calculados y no tener que volver a realizar las mismas operaciones. Tras estas mejoras y la consecución de los resultados mostrados el siguiente paso para conseguir una eficiencia aún mayor en tiempo de ejecución pasaba por la paralelización del algoritmo.

Tras el análisis del código secuencial podemos observar cómo la información más importante del software implementado viene dada por la proporción de tiempo que consume el proceso de interpolación con respecto al tiempo total de ejecución del algoritmo, estos resultados quedan mostrados en las figuras 8(a) y 8(b).

En estas figuras se ha representado ese porcentaje de tiempo con respecto al valor de la multiplicidad de los puntos para varios valores máximos del grado de y (dy), para un número de puntos diferente. La representación ha sido realizada mediante un promedio de 10 muestras temporales, en las figuras quedan representados también los valores máximo



(a) np=10 (b) np=20

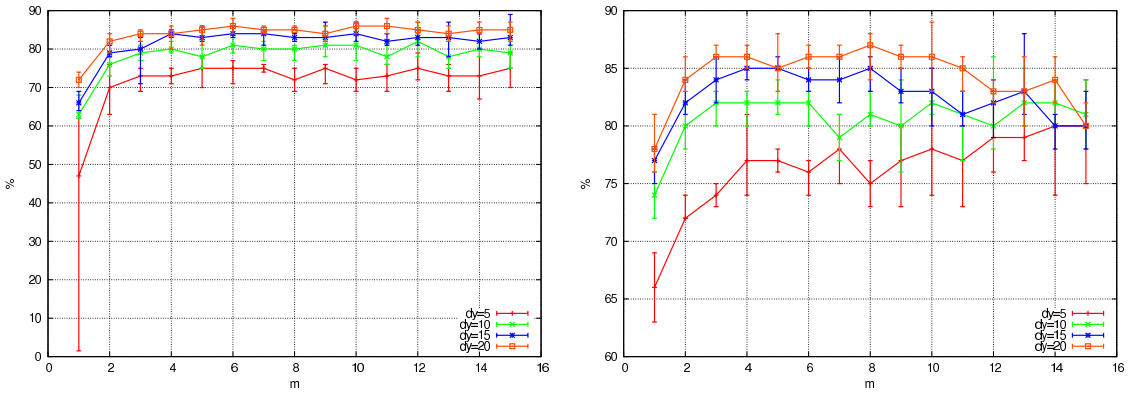
Figura 8: Proporción del tiempo de interpolación vs tiempo total de ejecución

y mínimo de las mediciones realizadas. En estas gráficas podemos observar cómo mayores valores de *dy* proporcionan un mayor tiempo de ejecución en el proceso de interpolación. Otra observación a destacar es la alta varianza que existe en los resultados obtenidos, es decir el tiempo es consumido por este paso del algoritmo tiene una fuerte dependencia con la instancia del problema. En cualquier caso, existe un alto porcentaje de tiempo de ejecución del algoritmo de decodificación que es consumido por el paso de interpolación.

De este modo es interesante desde un punto de vista de eficiencia centrar el estudio de la paralelización en esta parte del algoritmo.

Si analizamos el pseudocodigo del algoritmo de interpolación (Algoritmo 2) se observa como dentro de este paso existe una fuerte dependencia de datos entre las distintas iteraciones del algoritmo de Koetter implementado. Entre los distintos bucles externos al existir esa dependencia una paralelización de esa parte del algoritmo daría lugar a una incorrecta paralelización del bucle en cuestión. Sin embargo existe un bucle interno en el cual se calculan las discrepancias de los polinomios, con estas discrepancias se van haciendo las actualizaciones de los polinomios, en él cual cada iteración es completamente independiente del anterior. En este bucle para cada uno de los polinomios candidatos se busca la discrepancia que existe en esa iteración del algoritmo, el cálculo de esa discrepancia es independiente para cada uno de los polinomios y puede hacerse en paralelo para cada uno de ellos. En las figuras 9(a) y 9(b) se representa la proporción de tiempo que consume el bucle que puede ser paralelizado con respecto al tiempo total que se invierte en el proceso de interpolación.

Afortunadamente podemos observar como la proporción de tiempo de ejecución consumida por la parte de código que puede ser paralelizada de forma eficiente es sustancialmente superior al consumido por el resto, es decir el código que debe ser ejecutado de forma secuencial es relativamente rápido con respecto a este bucle interior. En estas gráficas se observa cómo ese ratio es prácticamente constante (con ratios entre el 75-85 %) con respecto a la multiplicidad m y al número de puntos np, obteniendo un valor más elevado conforme aumenta el valor de dy. Esto último era lógico de preveer que ocurriese,



(a) np=10 (b) np=20

Figura 9: Proporción del tiempo del bucle interno con respecto al tiempo total del paso de interpolación

debido que el tener un mayor valor de dy supone tener un aumento proporcional en el número de polinomios candidatos a que cumplan las condiciones de interpolación y de los cuales hay que calcular su discrepancia.

De nuevo, la parte del código que más tiempo de ejecución consume es susceptible de ser paralelizada lo cual puede potenciar una aceleración favorable en el cálculo de la decodificación, y es lógico pensar con los resultados obtenidos que será mas eficiente la paralelización para valores más elevados del parámetro dy. De cualquier modo la fracción secuencial del algoritmo limita el incremento de la velocidad que puede conseguirse en un programa utilizando múltiples procesadores. Por tanto la mejora que puede obtenerse tiene un máximo teórico el cual no podemos superar debido a esa fracción serie, ese valor teórico máximo de speedup puede ser calculado mediante la ley de Amdahl

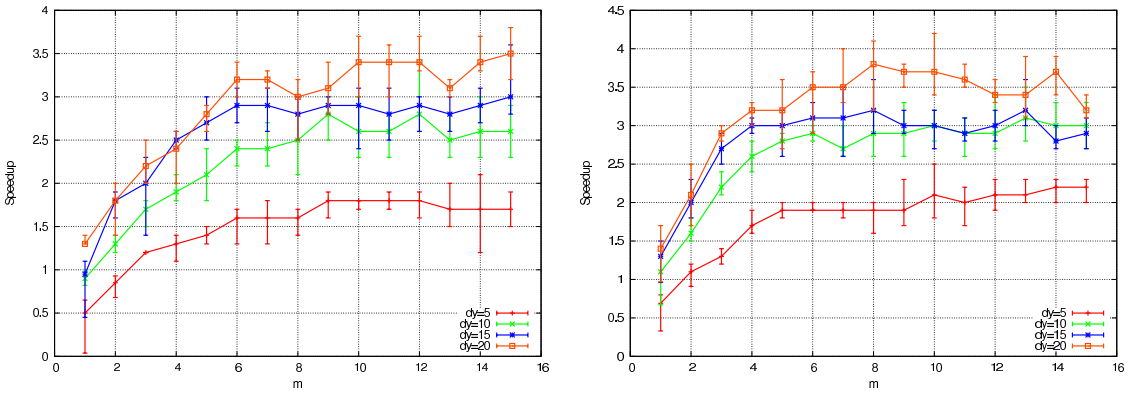
$$A = \frac{1}{(1-P) + \frac{P}{S}} \tag{61}$$

donde P es la fracción de tiempo que consume la parte la cual ha sido mejorada y S es el factor de mejora que se ha introducido en el subsistema mejorado, en nuestro caso corresponden a la fracción paralela y el número de procesadores respectivamente. Por tanto para calcular el máximo valor de speedup alcanzable cuando el número de procesadores tiende a ∞ tenemos que la ecuación (61) queda como $A = \frac{1}{1-P}$, anteriormente hemos extraído que la parte paralelizable consume entre el 75-85 % del tiempo total de ejecución, por lo que aplicando la fórmula anterior obtenemos que el incremento de velocidad o máximo speedup alcanzable se encuentra en el rango de $4-6,6$ debido a la ley de Amdahl

En primer lugar se pensó en implementar el algoritmo en el lenguaje de programación CUDA [29], sin embargo este lenguaje no encaja dentro de nuestro algoritmo debido a que solamente existe un bucle paralelizable y habría una clara infeficiencia en la paralelización, pues solo existen pocos hilos (dy es pequeña), por lo que el uso de este lenguaje sería deficiente.

Finalmente se optó por la utilización de directivas de compilación OpenMP para la

paralelización del bucle que calcula las discrepancias en el algoritmo de Koetter. Tras este paso se han obtenido los tiempos de ejecución del proceso de interpolación tanto para el código secuencial como para el paralelo. Con estos tiempos se han calculado los valores de speedup conseguidos los cuales se muestran en las figuras 10(a) y 10(b).



(a) np=10 (b) np=20

Figura 10: Speedup conseguidos en el proceso de interpolación

En las gráficas mostradas podemos observar como los valores de speedup obtenidos se encuentran en un rango comprendido entre 1,7 y 3,5 obteniendo distintos resultados dependiendo de los valores que tomen los parámetros en la ejecución, varían también en cada una de las simulaciones ejecutadas pues tiene una fuerte dependencia con la instancia del problema. Podemos hacer una comparación de los valores prácticos obtenidos con los valores máximos proporcionados por la ley de Amdahl y de este modo obtenemos una eficiencia sobre los resultados logrados que oscilan desde el 25 % hasta el 85 %.

5. Conclusiones y lineas futuras

De los capítulos presentados en este trabajo pueden ser extraídas una serie de importantes conclusiones. Los códigos Reed-Solomon son una familia de códigos ampliamente utilizados en multitud de estándares y aplicaciones y por ello es importante el estudio y desarrollo de implementaciones prácticas y eficientes en la decodificación de este tipo de códigos. En el presente proyecto se ha realizado una paralelización a nivel software de un algoritmo de decodificación de lista el cual mejora el rendimiento en decodificación con respecto a la decodificación única. En la bibliografía se pueden encontrar implementaciones hardware en FPGA del algoritmo de decodificación ([5],[38],[6]), sin embargo no se han encontrado hasta la fecha implementaciones y variaciones del algoritmo en el contexto de sistemas multinúcleos. Es por ello que este trabajo se enmarca dentro de este proposito teniéndo estos sistemas un amplio uso en dispositivos móviles. En la actualidad los dispositivos portátiles cada vez incorporan más sistemas multinúcleos por lo tanto una implementación paralela basada en este paradigma de programación multihilo resulta de interés para estos dispositivos.

De los resultados obtenidos podemos concluir como en el algoritmo existe una fuerte dependencia de datos entre las distintas iteraciones dentro del paso que consume la mayor parte de la ejecución. Es por ello que la paralelización solo puede ser realizada para el bucle más interno del algoritmo en el cual las iteraciones con completamente independientes. De este modo, tenemos una significativa fracción serie que nos limita el máximo speedup alcanzable siendo la razón por la cual se han conseguido unos resultados de eficiencia moderados.

En última instancia, el software implementado cumple la función de decodificador en un algoritmo de *fingerprinting* pero puede ser utilizado para otras aplicaciones en la cuales se utilice un código Reed-Solomon como codificación de canal. En nuestro caso particular se pretende utilizar este decodificador en un sistema MIMO con decodificación iterativa. La mejora en el rendimiento utilizando este tipo de decodificación es significativa debido a que ahora el decodificador no queda limitado por la distancia mínima. Utilizando la lista de posibles palabras código como información de realimentación entre el detector y el decodificador se pretende obtener una mejora significativa en la detección del sistema MIMO.

En cuanto a las líneas futuras de mejora del software se puede contemplar la posibilidad de paralelizar el paso de factorización el cual consume poco tiempo de ejecución, sin embargo esta paralelización no es determinista pues depende de como vaya evolucionando la obtención de las raíces. Solo serían paralelizables los procesos que aparecen en el mismo nivel del árbol de la figura 3, con un paradigma de paralelismo a nivel de tareas. Esta pobreza hace enfocar el interés en una posible paralelización "segmentada (pipeline)", decodificando varias palabras recibidas al mismo tiempo en etapas distintas. Ello requiere un estudio más profundo para poder realizar la segmentación, el equilibrio entre sus etapas,etc. y verificar que es posible obtener buenos resultados con este último paradigma de programación.

Otra posibilidad es el estudio e implementación de otros algoritmos que resuelvan el problema de interpolación y factorización y los cuales tengan un potencial más alto de paralelización pudiendo tener una mejora en el tiempo de de ejecución.

6. Agradecimientos

Este trabajo ha sido realizado gracias a la fnanciación del Vicerrectorado de Investación de la UPV a través del Programa de Apoyo a la Investigación y Desarrollo (PAID-05-11-2733) y a la Generalitat Valenciana a través del proyecto PROMETEO/2009/013.

Referencias

[1] G. Blakley y G. Kabatiansky A. Barg. Digital fingerprinting codes: Problems, statements, constructions, identification of traitors. *IEEE Transactions on information Theory*, 49, 2003.

[2] D. Augot and L. Pecquet. *An alternative to factorization: a speedup for Sudan´s decoding algorithm and its generalization to algebraic-geometric codes.* INRIA, Le Chesnay Cedex, Rapport de Recherche 3532, 1998.

[3] E. Berlekamp. *Algebraic Coding Theory.* McGraw Hill, 1968. New York (USA), 1968.

[4] M. Bossert. *Chanel Coding for Telecommunications.* West Sussex (UK): John Wiley & Sons, Ltd, 1999.

[5] Laurier Bouulianne and Warrer J. Gross. SIMD Implementation of Interpolation in Algebraic Soft-Decision Reed-Solomon Decoding. *Signal Processing Systems Design and Implementation*, pages 759–755, November 2005.

[6] Bainan Chen and Xinmiao Zhang. FPGA implementations of a factorization processor for soft-decision Reed-Solomon decoding. *Circuits and Systems*, pages 944–947, July 2006.

[7] S. Czapor, K. Geddes, and G. Labahn. *Algorithms for Computer Algebra.* Kluwer Academic, 1992.

[8] L Dagum. OpenMP: an industry standard API for shared memory programming. *Computacional Science & Engineering, IEEE*, pages 45–55, 1998.

[9] Ana M. de Viola-Prioli and Jorge E. Viola-Prioli. *Teoría de Cuerpos y Teoría de Galois.* Reverté, 2006.

[10] Peter Elias. List Decoding for noisy channels. Technical report, Laboratory of electronics massachusets institute of Technology, Septiembre 1957.

[11] R.J Mc Eliece. The Guruswami-Sudan Decoding Algotihm for Reed-Solomon Codes. Technical report, California institute of Technology, Pasadena, California, Mayo 2003.

[12] Sergei V. Fedorenko and Peter Trifonov. Finding roots of polynomials over finite fields. *IEEE Transactions on communications*, November 2002.

[13] M. Francis and R. Green. Forward Error Correction in Digital Television Broadcast System. *Xilinx WP270 v1.0*, 2007.

[14] Karypis-Kumar Grama, Gupta. *Introduction to Parallel Computing.* Addison-Wesley, 2003.

[15] T. Hoholdt and R. Nielsen. Decoding Hermitian codes with Sudan‘s algorithm. *Lecture Notes in Computer Science*, pages 260–270, 1999.

[16] T. Hoholdt and R. Nielsen. Decoding Reed-Solomon codes beyond half the minimum distance. *Coding Theory, Cryptography and Related Areas*, pages 221–236, 2000.

[17] L. Huguet and J. Rifá. *Comunicación Digital.* Masson, S.A. Barcelona (Spain), 1991.

[18] R. Koetter and A. Vardy. Algebraic soft-decision decoding of Reed-Solomon codes. *IEEE Transactions on Information Theory*, 1(11):2809–2825, 1999.

[19] R. Koetter and A. Vardy. A complexity reducing transformation in algebraic list-decoding of Reed-Solomon codes. *Procedings of the IEEE Information Thery Workshop*, 2003.

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

[21] Marcel Fernández Miguel Soriano A. González M. Simarro Haro, José Moreira and F.J Martínez Zaldívar. Parallelization of the interpolation process in the Koetter-Vardy soft-decision list decoding algorithm. *Proceeding of the 2012 International Conference on Computacional and Mathematical Methods in Science and Engineering*, pages 1102–1110, July 2012.

[22] F. MacWilliams and N. Sloane. *The Theory of Error-Correcting Codes*. Amsterdam (Holland): North-Holland, 1997.

[23] Todd K. Moon. *Alternate Decoding Algorithms for Reed-Solomon Codes*. Wiley, 2005.

[24] Maria Ángeles Simarro Haro. *Proyecto Final de Carrera: Decodificación de lista de códigos Reed-Solomon*. Universidad Politécnica de Valencia, 2011.

[25] A. Vardy R. Koetter, J. Ma. and A. Ahmed. Efficient interpolation and factorization in algebraic soft-decision decoding of Reed-Solomon codes. *Proceedings of the IEEE International Symposium on Information Theory*, pages 365–365, 2003.

[26] Dave Kohr Dror Maydan Jeff McDonald Ramesh Menon Rohit Chandra, Leonardo Dagum. *Programming in OpenMP*. Academic Press, 2001.

[27] R. Roth and G. Ruckenstein. Efficient decoding of Reed-Solomon codes beyond half the minimun distance. *IEEE Transactionson Information Theory*, 46(1):246–257, 2000.

[28] A. Cabrera J. Suardiaz S. Lujan, S. Almagro and F. Cerdan. Códigos RS y su aplicación a la capa física 802.16 en FPGAs. *III Jornadas de Introducción a la Investigación de la UPCT*.

[29] Jason Sanders and Edward Kandrot. *Cuda by example An Introduction to General-Purpose GPU Programming*. Addison-Wesley, 2010.

[30] M. Shokrollahi and H. Wasserman. List decoding of algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45(2):432–437, 1999.

[31] M. Soriano, M. Fernández, E. Sayrol, J. Tomás, J. Casanellas, J. Peguerolesand, and J. Hernández-Serrano. Multimedia copyright protection platform demostrador. *Lecture Notes in Computer Science*, 3477:411–414, 2005.

[32] Vijay K. Bhargava Stephen B. Wicker. *Reed-Solomon Codes and their Applications*. Wiley, 1999.

[33] M. Sudan. Decoding of Reed-solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.

[34] J. van Lint. *Introduction to Coding Theory*. Heidelberg (Germany): Springer-Verlang, 1982.

[35] Madhu Sudan Vankatesan Guruswami. Improved decoding of reed-solomon codes and algebraic geometric codes. *IEEE Transactions on Information Theory*, 45(6):1757–1797, 1999.

[36] Madhu Sudan Vankatesan Guruswami. Improved decoding of Reed-Solomon codes and algebraic geometric codes. *IEEE Transactions on Information Theory*, 45(11):175–989, October 1999.

[37] R. Koetter W. Gross, F. Kxchischang and P.G. Gulak. Applications of algebraic soft-decision decoding of Reed-Solomon codes. *Submited as a transactions paper to the IEEE Transactions on Communications*, 2003.

[38] Frank R. Kschischang Warren J. Gross 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*, March 2007.

[39] Ralf Koetter Warren J. Gross, Frank R. Kschischang and P. Glenn Gulak. Simulation results for algebraic soft-decision decoding of Reed-solomon codes. *Proceedings of the 21'st Biennial*.

[40] R. Koetterand P.G Gulak W.Gross, F.Kschinchang. Aplications of algebraic soft-decision decoding of Reed-Solomon codes. *Submited as a transacitons paper to the IEEE Transactions on Communications*, 2003.

[41] X.-W. Wu and P. Siegel. Efficient root-finding algorithm with application to list decoding of algebraic-geometric codes. *IEE Transactions on Information Theory*, 47(6):2579–2587, 2001.

[42] M. Fernández y M. Soriano. Fingerprinting concatenated codes with efficient identification. *Lecture Notes in Computer Science*, 2433:459–470, 2002.

[43] M. Sudan y V. Guruswami. Improved decoding of Reed-Solomon and algebraic-geometry codes. *Information Theory, IEEE Transactions*, 45:1757 – 1767, Septiembre 1999.

A. Artículos

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

 ISBN:978-84-615-5392-1

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 $\mathrm{RS}[n,k]$, is defined as the following vector subspace of $GF(q)^n$:

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

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.

 ISBN:978-84-615-5392-1

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 $(x_i, y_i, m_i) \in S$
  for $(r, s) = (0, 0)$ to $(m_i - 1, 1)$
    for $j = 0$ to dy
      DISC: $\Delta_j = \text{coeff}(Q_j(x + x_i, y + y_i), x^r\ y^s)$
    $j^* = \text{argmin}_{(1,k-1)}\{Q_j : j \in J\}$
    $f = Q_j$
    $\Delta = \Delta_{j^*}$
      if $(j \neq j^*)$
        $Q_j = \Delta Q_j - \Delta_j f$
      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 $\text{coeff}(Q(x, y), x^r\ y^s)$ 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

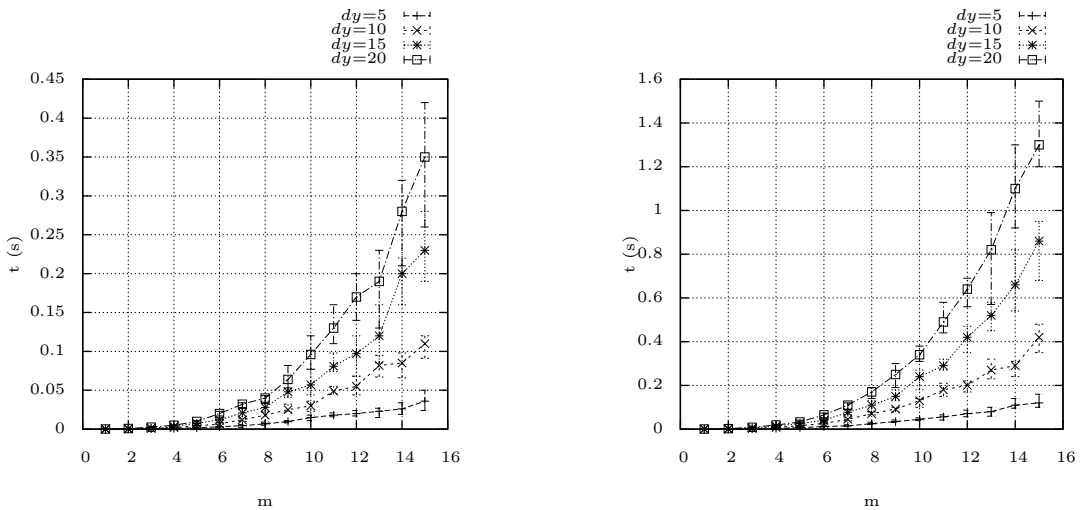
All the tests have been executed in a SMP dual Intel(R) Xeon(TM) Hex-core CPU X5675 @3.07 GHz with 128 GB of main memory, without multithreading, running a Linux operating system with kernel 2.6.32, using the Intel `icc` C/C++ compiler version 12.1.

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.

 ISBN:978-84-615-5392-1

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) $n_p = 20$

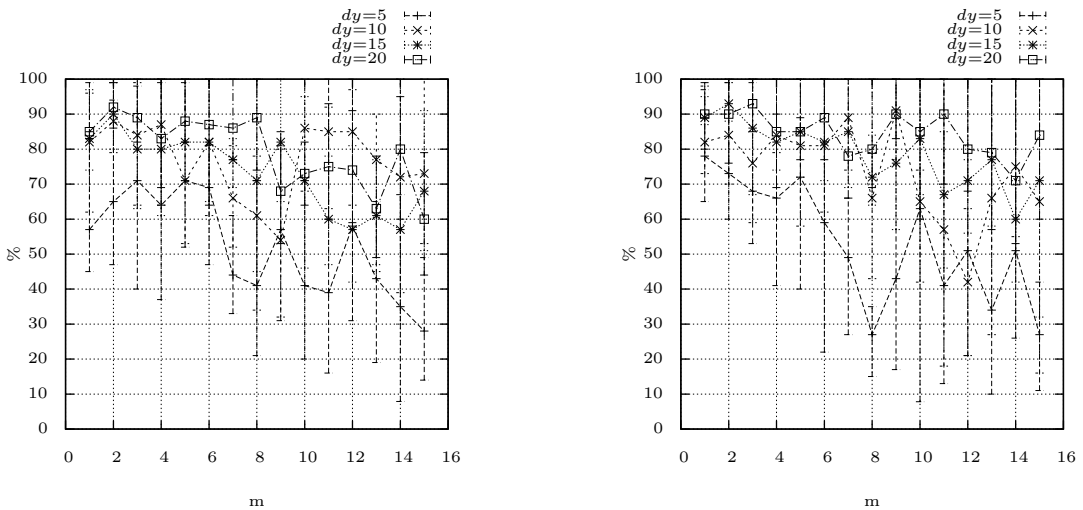
Figure 1: Sequential interpolation execution time

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

 ISBN:978-84-615-5392-1



(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

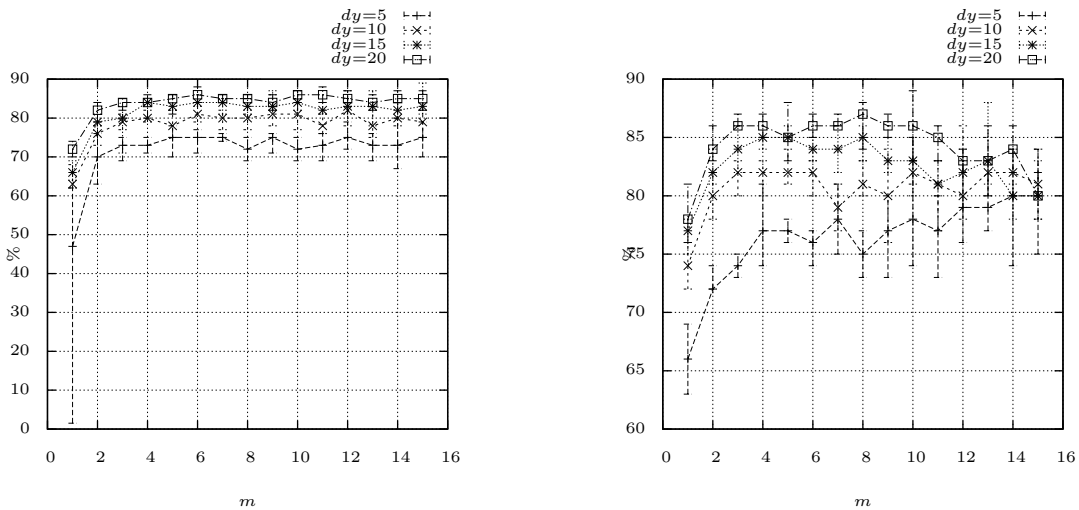
We have used *OpenMP* compiler directives to parallelize the innermost loop of the Koetter algorithm. Parallel and sequential times have been obtained for the interpolation process using the `-O3` compiler switch for all cases.

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

 ISBN:978-84-615-5392-1



(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.

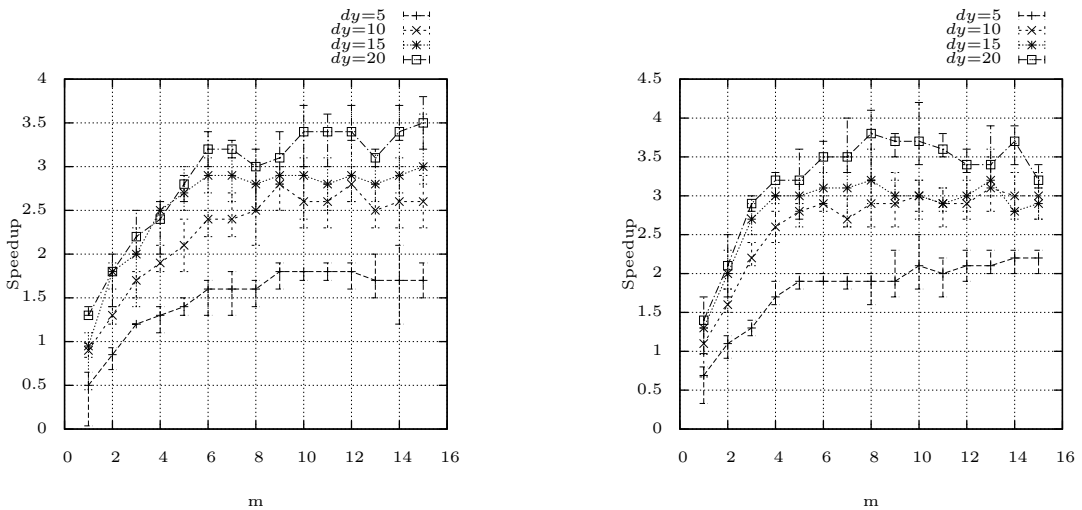
Acknowledgements

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.

 ISBN:978-84-615-5392-1



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

Figure 4: Speedup in Koetter interpolation algorithm

[2] M. Sudan, "Decoding of Reed-Solomon codes boyond de error correction bound", *J. Complesity*, 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*

 ISBN:978-84-615-5392-1

[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 algebraicgeometric 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.

 ISBN:978-84-615-5392-1