Application Report
ZHCA579 – November 2013

CRS 编译码原理和在 TI C6000 DSP 上的优化实现

James Li, Wei Chen
Multi-core DSP / FAE

摘要

CRS 编码的全称是 Cauthy Reed-Solomon Codes。它是一种擦除码（Erasure Coding）编码方式，常用于数据冗余应用，常用的 RAID6 使用的算法就是一种简化的 RS 编码。RS 编译码需要做伽罗华域运算，并且编码矩阵和译码矩阵的构造都涉及了许多线性代数知识，对初学者比较晦涩难懂。所以本文从工程应用的角度以大量实例结合示意图介绍了这种方法。本文还用 TI C6000 DSP 专有的伽罗华域乘法指令对 RS 编译码函数做了优化和性能分析。希望对使用 TI DSP 实现相关应用的读者有帮助。

文档修改纪录

Version	Date	Author	Notes
0.1	July 2013	James Li, Wei Chen	First release
1.1	Sept 2013	James Li, Wei Chen	*Updates*
1.2	Oct 2013	James Li, Wei Chen	*Updates*

目录

表

图

1 RS 编码简介

CRS 编码的全称是 Cauthy Reed-Solomon Codes，它是一种擦除码（Erasure Coding）编码方式，常用于数据冗余应用，常用的 RAID6 使用的算法就是一种简化的 RS 编码。在使用 RS 编码冗余的存储系统中，通过 RS 编码生成校验数据。如果在存储和传输期间数据发生有限程度的损坏，就可以把剩下的有效数据和正确的校验数据联合起来做 RS 译码，恢复出原始数据。需要注意的是，RS 译码的时候是已知哪些数据包和校验包是正确的。这是 RS 编码和信道编码的重要差异之一。关于 RS 编码的更精确的描述是：

假设原始数据长度是 $L*M$ 。编码的时候把原始数据均分成 M 个包，每个包的长度是 L 。需要构造的编码矩阵尺寸是 $R*M$ ，其中 R 是校验包的个数（例如典型的配比是 $R=3,M=9$ ）。编码的过程是用 $R*M$ 的编码矩阵乘以 $M*L$ 的数据矩阵得到 $R*L$ 的校验矩阵，校验矩阵是 R 个长度为 L 的校验包。在存储和传输的过程中：

1) M 个数据包和 R 个校验包都被存储和传输。如果超过 R 个包被损坏那原始数据就无法恢复。
2) 如果有 R 个包被损坏，且损坏的都是校验包，那么 M 个数据包全部正确，不需要做译码就可以恢复出原始数据。
3) 如果有 R 个包被损坏，其中 R' 个是数据包（有 $R'<R$ ），那么可以通过 CRS 译码恢复出原始数据。做法是：
   a) 选取所有正确的数据包再加上任意 R' 个正确的校验包（总共 M 个向量），组成一个 $M*L$ 的矩阵。
   b) 根据选取的 M 个向量对应的编码矩阵计算译码矩阵（计算过程就是矩阵求逆，然后从逆矩阵中提取出译码矩阵），译码矩阵的尺寸是 $R'*M$ 。
   c) 用译码矩阵乘与步骤 a)构造的 $M*L$ 矩阵得到 $R'*L$ 的矩阵。这个 $R'*L$ 的矩阵就是恢复出来的 R' 个被损坏的数据包。

上面所述的生成编译码矩阵以及编码译码过程的运算都是伽罗华域的运算，所以本文先阐述伽罗华域的运算规则，然后详细说明 RS 编译码的每个步骤，最后介绍在 TI C6000 DSP 上 RS 编译码函数的优化实现。

2 伽罗华域的基本运算

伽罗华域又被称为 Galois 域或有限域，通常表示成 $GF(2^L)$ 。伽罗华域的元素个数是有限的。 $GF(2^L)$ 有 2^L 个元素: $0,1,2,3,.....2^L-1$ 。例如 $GF(2^3)$ 有 8 个元素，分别是 0,1,2,3,4,5,6,7。应用中最常见的是 $GF(2^8)$ ，因为它的有效元素区间是 $0,1,2,3,.....2^8-1$ ，刚好是一个字节的表示范围。本文讨论的 CRS 编码译码运算都是基于 $GF(2^8)$ 的。 $GF(2^L)$ 域的基本运算定义如下。

2.1 加减运算

$GF(2^L)$ 的加法和减法是等效的，就是二进制比特位的异或操作。

2.2 乘法运算

$GF(2^L)$ 的乘法和实数域的乘法很类似，只是在乘积超出有限域的范围时要用生成多项式取模（也就是减去生成多项式）。下表是对应 $GF(2^1),GF(2^2),GF(2^3),.....GF(2^{15})$ 的生成多项式。需要注意的是生成多项式不是唯一的，例如 $GF(2^8)$ 有多个生成 多项式：0x11D， 0x12B， 0x12D 等。

```
{        0x3,   0x7,   0xB,   0x13,   0x25,  0x43,    0x83,
0x11D, 0x211, 0x409, 0x805, 0x1053, 0x201B, 0x402B, 0x8003}
```

下面是 $GF(2^3)$ 的乘法运算示例， $GF(2^3)$ 的生成多项式是 0xB。

```
  6*1 = 6 |  6*2 = 7 |  6*3 = 1 |  6*4 = 5 |  6*5 = 3 |  6*6 = 2 |  6*7 = 4 |
    110   |    110   |    110   |    110   |    110   |    110   |    110   |
    001   |    010   |    011   |    100   |    101   |    110   |    111   |
    ----  |    ----  |    ----  |    ----  |    ----  |    ----  |    ----  |
    110   |    000   |    110   |    000   |    110   |    000   |    110   |
   000    |   110    |   110    |   000    |   000    |   110    |   110    |
 ^000     | ^000     | ^000     | ^110     | ^110     | ^110     | ^110     |
  ------  |  ------  |  ------  |  ------  |  ------  |  ------  |  ------  |
  00110   |  01100   |  01010   |  11000   |  11110   |  10100   |  10010   |(1)乘累加结果
          |  ^1011   |  ^1011   | ^1011    | ^1011    | ^1011    | ^1011    |
  ------  |  ------  |  ------  |  ------  |  ------  |  ------  |  ------  |
    110   |    111   |    001   |   1110   |   1000   |    010   |    100   |(2)减生成多项式
                                | ^1011    | ^1011    |          |          |
                                | ------   | ------   |          |          |
                                |    101   |    011   |          |          |(3)减生成多项式
```

(1)计算乘累加的结果
(2)因为乘累加结果超出了有限域的范围，所以要从最高位开始和生成多项式异或(减去生成多项式)
(3)部分结果要需要减多次生成多项式

$GF(2^L)$ 的运算可以用查表的方法实现，例如下面是 $GF(2^3)$ 的加法表和乘法表（摘自参考文献【2】）

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

Addition

*	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

Multiplication

图 1 GF(2^3)的加法和乘法表

2.3 除法运算

$GF(2^L)$ 域的除法可以转化成乘法。因为 $A/B = A*\frac{1}{B}$ ，所以可以先查表找到 B 的倒数，然后再用乘法计算 $A*\frac{1}{B}$ 。例如在 $GF(2^3)$ ，0 没有倒数，{1,2,3,4,5,6,7}的倒数分别是{1,5,6,7,2,3,4}。

2.4 二的幂次运算

参考文献【1】中给出一种把 $GF(2^L)$ 域的乘除操作转换成二的幂次加减操作的方法。用这种方法计算简单而且可以大大缩减查找表的大小。用这种方法实现 $GF(2^L)$ 域的乘除运算只需要两个大小为 2^L 个元素的查找表。以 $GF(2^8)$ 为例，幂次到元素的查找表（ExptoFe）的作用是：用索引 x 查这个表就可以得到在 $GF(2^8)$ 域上 2^x 对应的元素。这个表存储的是 $GF(2^8)$ 上 [2^0, 2^1, 2^2, 2^3,.. 2^255] 对应的元素：

```
  1,   2,   4,   8,  16,  32,  64, 128,  29,  58, 116, 232, 205, 135,  19,  38,
 76, 152,  45,  90, 180, 117, 234, 201, 143,   3,   6,  12,  24,  48,  96, 192,
157,  39,  78, 156,  37,  74, 148,  53, 106, 212, 181, 119, 238, 193, 159,  35,
 70, 140,   5,  10,  20,  40,  80, 160,  93, 186, 105, 210, 185, 111, 222, 161,
 95, 190,  97, 194, 153,  47,  94, 188, 101, 202, 137,  15,  30,  60, 120, 240,
```

```
253, 231, 211, 187, 107, 214, 177, 127, 254, 225, 223, 163,  91, 182, 113, 226,
217, 175,  67, 134,  17,  34,  68, 136,  13,  26,  52, 104, 208, 189, 103, 206,
129,  31,  62, 124, 248, 237, 199, 147,  59, 118, 236, 197, 151,  51, 102, 204,
133,  23,  46,  92, 184, 109, 218, 169,  79, 158,  33,  66, 132,  21,  42,  84,
168,  77, 154,  41,  82, 164,  85, 170,  73, 146,  57, 114, 228, 213, 183, 115,
230, 209, 191,  99, 198, 145,  63, 126, 252, 229, 215, 179, 123, 246, 241, 255,
227, 219, 171,  75, 150,  49,  98, 196, 149,  55, 110, 220, 165,  87, 174,  65,
130,  25,  50, 100, 200, 141,   7,  14,  28,  56, 112, 224, 221, 167,  83, 166,
 81, 162,  89, 178, 121, 242, 249, 239, 195, 155,  43,  86, 172,  69, 138,   9,
 18,  36,  72, 144,  61, 122, 244, 245, 247, 243, 251, 235, 203, 139,  11,  22,
 44,  88, 176, 125, 250, 233, 207, 131,  27,  54, 108, 216, 173,  71, 142,  1
```

元素到幂次的查找表（FetoExp）的作用是：用索引 y 查这个表就可以得到在 $GF(2^8)$ 域上元素 y 对应的二的幂次。这个表的内容是：

```
#,   0,   1,  25,   2,  50,  26, 198,   3, 223,  51, 238,  27, 104, 199,  75,
  4, 100, 224,  14,  52, 141, 239, 129,  28, 193, 105, 248, 200,   8,  76, 113,
  5, 138, 101,  47, 225,  36,  15,  33,  53, 147, 142, 218, 240,  18, 130,  69,
 29, 181, 194, 125, 106,  39, 249, 185, 201, 154,   9, 120,  77, 228, 114, 166,
  6, 191, 139,  98, 102, 221,  48, 253, 226, 152,  37, 179,  16, 145,  34, 136,
 54, 208, 148, 206, 143, 150, 219, 189, 241, 210,  19,  92, 131,  56,  70,  64,
 30,  66, 182, 163, 195,  72, 126, 110, 107,  58,  40,  84, 250, 133, 186,  61,
202,  94, 155, 159,  10,  21, 121,  43,  78, 212, 229, 172, 115, 243, 167,  87,
  7, 112, 192, 247, 140, 128,  99,  13, 103,  74, 222, 237,  49, 197, 254,  24,
227, 165, 153, 119,  38, 184, 180, 124,  17,  68, 146, 217,  35,  32, 137,  46,
 55,  63, 209,  91, 149, 188, 207, 205, 144, 135, 151, 178, 220, 252, 190,  97,
242,  86, 211, 171,  20,  42,  93, 158, 132,  60,  57,  83,  71, 109,  65, 162,
 31,  45,  67, 216, 183, 123, 164, 118, 196,  23,  73, 236, 127,  12, 111, 246,
108, 161,  59,  82,  41, 157,  85, 170, 251,  96, 134, 177, 187, 204,  62,  90,
203,  89,  95, 176, 156, 169, 160,  81,  11, 245,  22, 235, 122, 117,  44, 215,
 79, 174, 213, 233, 230, 231, 173, 232, 116, 214, 244, 234, 168,  80,  88, 175,
```

把 $GF(2^L)$ 域的乘除操作转换成二的幂次加减操作的方法举例如下：

FetoExp[5] = 50 说明在 $GF(2^8)$ 有 $2^{50}=5$，可以在 ExptoFE 表中验证 ExptoFE[50]=5。

FetoExp[7] = 198 说明在 $GF(2^8)$ 有 $2^{198}=7$，可以在 ExptoFE 表中验证 ExptoFE[198]=7。

根据两个数相乘等于它们的幂次相加的特性，在 $GF(2^8)$ 有 $5*7=2^{50}*2^{198}=2^{248}$，查表有 ExptoFE[248]=27，所以有在 $GF(2^8)$ 上 5*7=27。可以手工计算验证如下：

```
     101
     111
---------
     101
    101
  ^101
---------
   11011
```

$GF(2^L)$ 域的求倒运算也可以通过转换成幂次来操作，例如：FetoExp[5] = 50 说明在 $GF(2^8)$ 有 $2^{50}=5$。因为 2^0 = 1，所以 5 的倒数的幂次 x 满足 (50+x)%255=0，有 x=205，ExptoFE[205]= 167，可以手工计算验证如下：

```
10100111
     101
---------
10100111
00000000
```

```
 ^10100111
 ------------
  1000111011
 ^100011101
 ------------
           1
```

3 CRS 编译码的原理

3.1 生成编码矩阵

CRS 编码的生成矩阵是 $G=[I \mid A]$，其中 I 是 $M*M$ 的单位阵， A 是一个 $R*M$ 的柯西矩阵。生成矩阵有一个重要特性： G 的任意一个 $M*M$ 子矩阵都是可逆的。柯西矩阵是以法国数学家柯西的名字命名的。它是定义在有限域的矩阵（可以不是方阵）。主要特性是：柯西矩阵的每个子矩阵都是柯西矩阵。所有柯西方阵都是可逆的，而且柯西矩阵求逆的运算量比常规矩阵小很多。在 $GF(2^L)$ 上构造柯西矩阵的步骤是：

步骤 1，构造元素 $X=\{x_1,x_2,...x_{R-1},x_R\}$ 和 $Y=\{y_1,y_2,...y_{M-1},y_M\}$，其中 x_i 是整数 i-1 的二进制表示。元素 y_i 是整数 $2^{L-1}+i-1$ 的二进制表示。 x_i 和 y_i 是 $GF(2^L)$ 上互相独立的元素，即 $X \cap Y=\varnothing$。

步骤 2，用上面得到的 x 和 y 元素构造下面的 $R*M$ 矩阵，就是柯西矩阵。

$$\begin{bmatrix} \frac{1}{x_1+y_1} & \frac{1}{x_1+y_2} & \frac{1}{x_1+y_3} & \cdots & \frac{1}{x_1+y_{M-1}} & \frac{1}{x_1+y_M} \\ \frac{1}{x_2+y_1} & \frac{1}{x_2+y_2} & \frac{1}{x_2+y_3} & \cdots & \frac{1}{x_2+y_{M-1}} & \frac{1}{x_2+y_M} \\ \cdots & . & & & & \\ \frac{1}{x_R+y_1} & \frac{1}{x_R+y_2} & \frac{1}{x_R+y_3} & \cdots & \frac{1}{x_R+y_{M-1}} & \frac{1}{x_R+y_M} \end{bmatrix}$$

注意要保证 X 和 Y 没有交集必须满足 $M<2^{L-1}$ 和 $R<2^{L-1}$。也就是当 $GF(2^L)$ 的 L =8 时，要求 $M<128$ 和 $R<128$。CRS 编码的应用完全满足这个条件。下面是两个 $GF(2^8)$ 的柯西矩阵：

```
2 x 4 的柯西矩阵
27 84 161 29
84 27 29 161
```

```
8 x 8 的柯西矩阵
27 84 161 29 124 204 228 176
84 27 29 161 204 124 176 228
161 29 27 84 228 176 124 204
29 161 84 27 176 228 204 124
124 204 228 176 27 84 161 29
204 124 176 228 84 27 29 161
228 176 124 204 161 29 27 84
176 228 204 124 29 161 84 27
```

3.2 CRS 编码

得到了生成矩阵就可以开始 CRS 编码。CRS 编码就是用 $R*M$ （需要注意的是：生成矩阵 $G=[I \mid A]$ 的尺寸是 $(M+R)*M$，因为上半部分 $M*M$ 是一个单位阵，所以缩减为 $R*M$ 的编码矩阵）的编码矩阵乘与 $M*L$ 的数据矩阵得到 $R*L$ 的校验矩阵。矩阵相乘是由乘加运算组成的。这些乘加运算都是 $GF(2^8)$ 上的乘加运算。下图是 CRS 编码的过程，参数是 M=4， R=2。

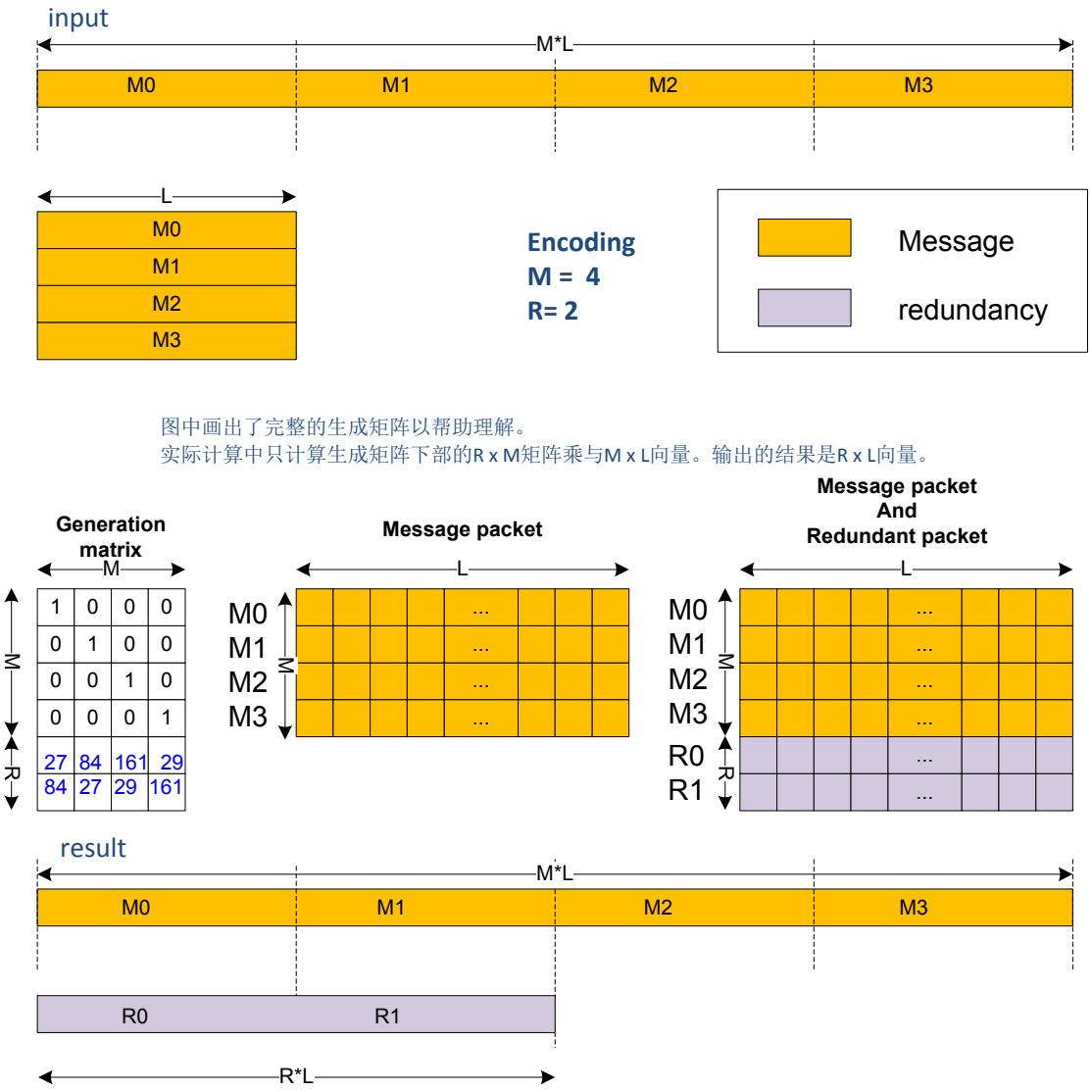

图 2 CRS 编码示意图

C 代码表示的 CRS 编码的过程如下。

```
#define M  (4)
#define R  (2)

char CuthyMat[R][M]={
27, 84, 161,  29,
84, 27,  29, 161};
char Msg[M][L];
char Redundancy[R][L];

void CRS_Enc()
{
    int i, j, k;
    memset(Redundancy,0,sizeof(Redundancy));
    for( i = 0; i < R; i++ )
    {
```

```
        for( j = 0; j < L; j++ )
        {
            for( k = 0; k < M; k++ )
            {
                //这里的乘号表示的是伽罗华域 GF(2^8)的乘法
                Redundancy[i][j] ^= Msg[k][j]*CuthyMat[i][k];
            }
        }
    }
}
```

可以看出，编码过程是大量的 $GF(2^8)$ 域的乘加运算，需要执行 $GF(2^8)$ 域乘法和加法各 $L*M*R$ 次。因为输入编码的数据长度是 $L*M$，所以在待编码的数据长度不变的情况下，CRS 编码的运算量是和 R 成正比的。

3.3 生成译码向量和译码

假设原有 M 个数据包，CRS 编码生成了 R 个校验包。在存储和传输的过程中，如果有小于等于 R 个包被损坏，其中 R' 个是数据包（有 $R' < R$），那么可以通过 CRS 译码恢复出原始数据。需要注意的是，译码的时候是已知哪些数据包和校验包是正确的，这是 RS 译码和信道译码的重要差异之一。CRS 译码的做法是：

1. 选取所有正确的数据包再加上 R' 个校验包（总共 M 个向量），组成一个 $M*L$ 的矩阵。
2. 这 M 个向量的每一个都对应生成矩阵中的一行。所以可以从生成矩阵中抽出这 M 行，组成一个 $M*M$ 的矩阵，称之为子矩阵 B。因为生成矩阵 $G=[I\,|\,A]$ 的任何一个 $M*M$ 子矩阵都是可逆的，所以可以求出 B 的逆矩阵，用 B^{-1} 和第一步得到的 $M*L$ 的矩阵相乘就可以恢复出所有 M 个数据包。当然正确接收的数据包就不需要再算了，所以可以简化成 $R'*M$ 的译码矩阵乘与 $M*L$ 的矩阵的乘法。可以恢复出 R' 个被损坏的数据包。

矩阵求逆有很多方法，根据 Cramer 法则做矩阵求逆的方法是：

$$A^{-1}=\frac{1}{\det(A)}(C^T)_{i,j}=\frac{1}{\det(A)}(C)_{j,i}=\frac{1}{\det(A)}\begin{bmatrix}C_{1,1} & C_{2,1} & \dots & C_{n,1}\\ C_{1,2} & C_{2,2} & \dots & C_{n,2}\\ \dots & \dots & \dots & \dots\\ C_{1,n} & C_{2,n} & \dots & C_{n,n}\end{bmatrix}$$

其中：$\det(A)$ 是矩阵 A 的特征值（deteminante，关于矩阵特征值的计算请参见线性代数的参考资料）。C 是 A 的伴随矩阵。C 中的每个元素 $C_{i,j}$ 是矩阵 A 在位置 (i,j) 的余数因子（Cofactor）。$C_{i,j}$ 是由矩阵在位置 (i,j) 的子行列式（minor）算出来的。子行列式 $M_{i,j}$ 表示把矩阵 A 去除第 i 行和第 j 列以后得到的小矩阵的特征值。例如：A 是一个 $3*3$ 的矩阵。

$$M_{2,3}=\det\begin{bmatrix}1 & 4 & \#\\ \# & \# & \#\\ -1 & 9 & \#\end{bmatrix}=\det\begin{bmatrix}1 & 4\\ -1 & 9\end{bmatrix}=1*9-(-1)*4=13$$

$$C_{2,3}=(-1)^{2+3}(M_{2,3}=-13)$$

参考资料[1]给出了计算柯西矩阵的 $C_{i,j}$ 的方法，计算复杂度是 $O(n^2)$，其中 n 是方阵的边长。柯西矩阵求逆的计算量远小于常规矩阵求逆的运算量。因为 M 个数据包和 R 个校验包损坏的组合是随机的，所以译码向量通常要动态计算。当然也可以用枚举的方法遍历所有的丢包场景，把所有译码向量都生成并存储。下图所示的例子 M=4,

R=2，数据包 M2 和 M3 被损坏，再经过译码被恢复出来。译码和编码的计算流程是完全相同的，所以实现的时候可以使用同一套代码。

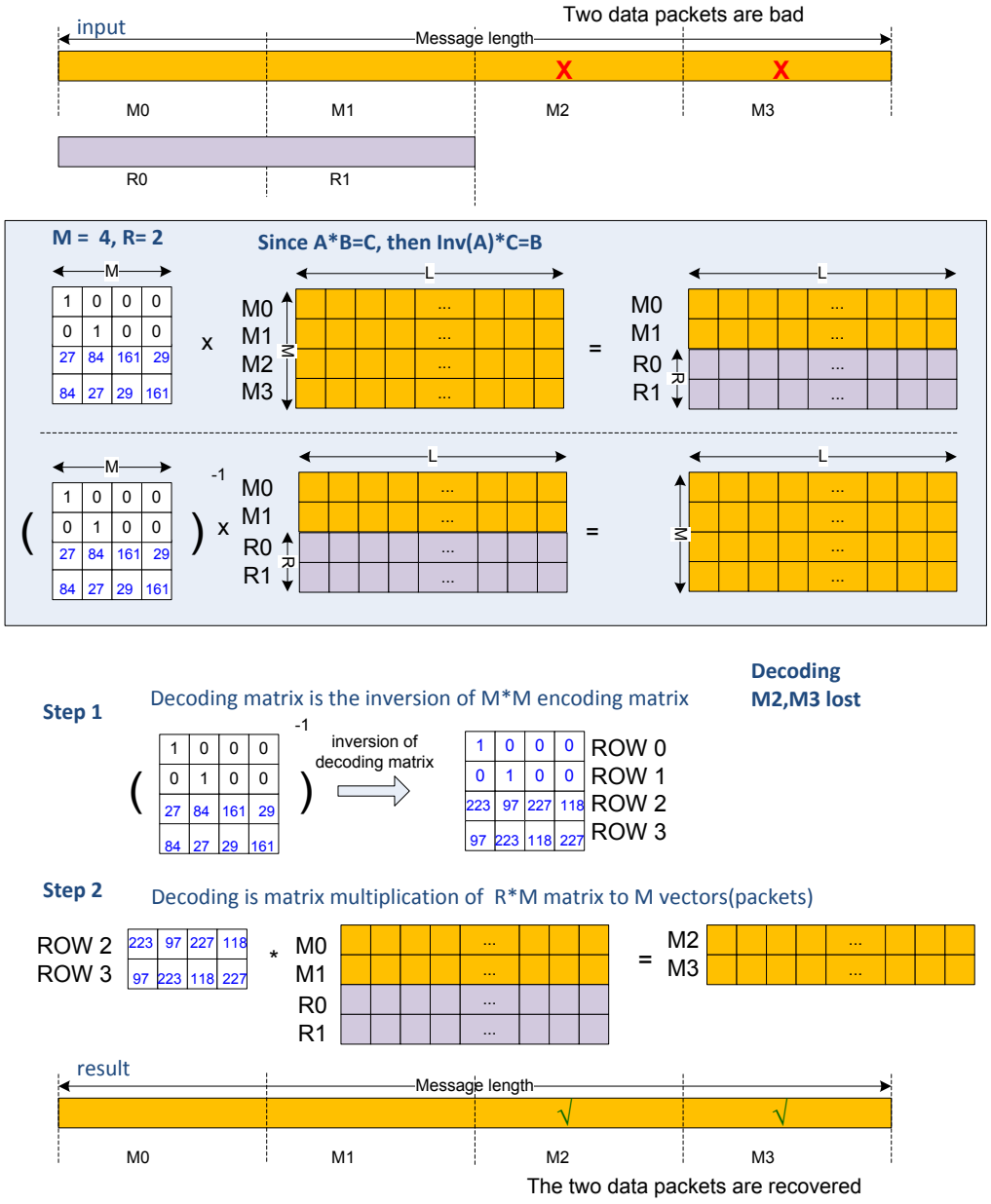

图 3 CRS 译码示意图

4 TI C6000 DSP 上编译码函数的优化实现

从上面的分析可以看出生成编码矩阵和计算译码矩阵的运算量都很小。运算量最大的是编码和译码的过程。通用 CPU 没有专门的硬件指令加速 $GF(2^8)$ 域的乘法，虽然可以用查表的方法实现，但是效率不高。然而 TI 的 C6000 DSP 有专门的指令 GMPY4，可以在 1 个 cycle 计算 8 个 $GF(2^8)$ 域的乘法。所以编译码运算在 DSP 实现非常高效。本文以 $M=8, R=8$ 为例编码实现了基于 C6000 DSP 的 CRS 编码函数，下面是实现细节。

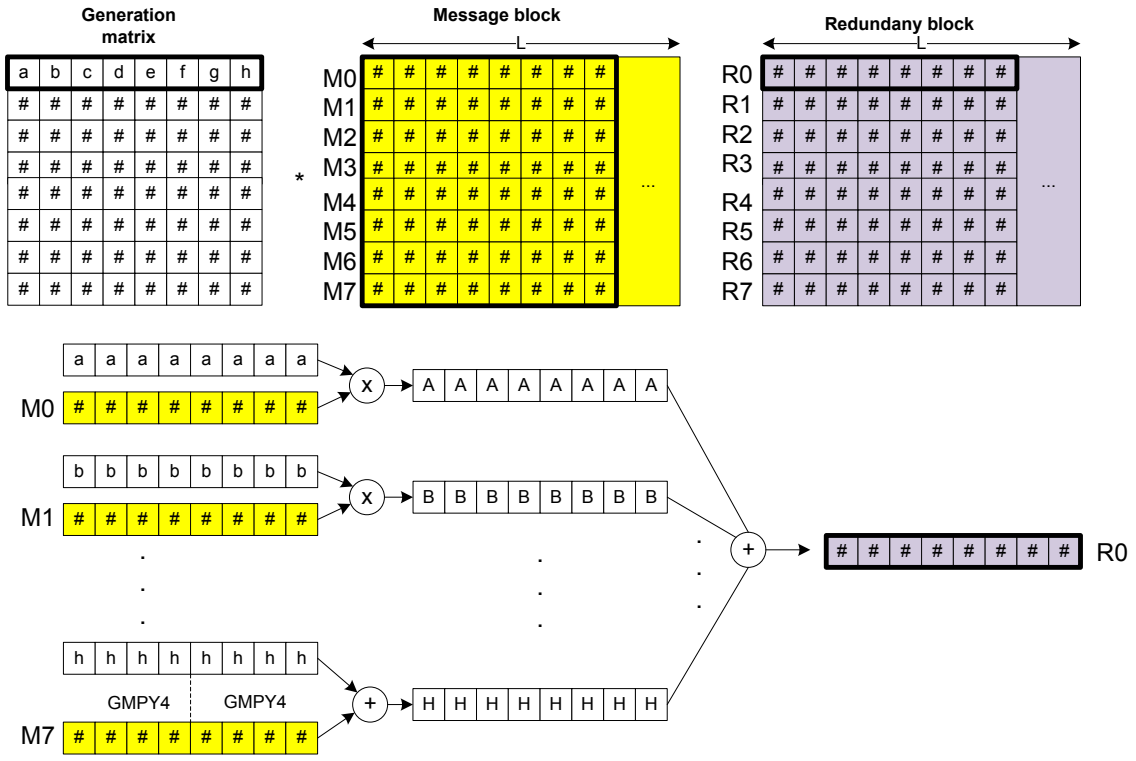

图 4 DSP 优化代码处理流程示意图

TI C6000 DSP核（C64，C64+，C66）都支持GMPY4指令，每条指令每个cycle可以计算4个 $GF(2^8)$ 域的乘法。这些DSP核的结构相近，由8个功能单元构成，分别是2个M，2个S，2个L和2个D。D单元用来存取数据，每个D单元的存取宽度是64比特。L和S单元用来做加减移位，布尔运算等算术逻辑运算等。M单元做乘法。GMPY4是在M单元执行的。因为有两个M单元，所以每个cycle可以计算8个 $GF(2^8)$ 域乘法。本文的实现中，内层循环每次计算如图1x8向量乘与8x8矩阵生成1x8向量。几个优化技巧是：

1) 使用intrinsic amem8加载待编码数据，amem8对应汇编指令LDDW，这是对齐加载指令并且每次加载64比特。这样可以最大限度发挥D单元的能力。每条指令可以加载8个M元素。
2) 生成矩阵的1x8向量存储在寄存器中（因为这些系数要被重复使用）以减轻D单元的压力。以M0为例，如图，M0的所有数据要和元素a相乘，因为一次加载了8个M0元素，所以生成矩阵的1x8向量的元素a也要被重复8次以便于GMPY4指令的并行处理。生成矩阵的1x8向量的8个元素都重复8次需要占用8*8/4=16个寄存器。C64以上版本的DSP核都有64个32bit寄存器，所以寄存器压力并不大。在代码中，重复过的生成矩阵元素存储在数组crsOperator[]中。从下面的C代码看，生成矩阵的元素也是从内存中加载的。但是在编译的时候如果使能了O3优化，编译器会自动把它们加载到寄存器中重复使用，可见C6000的编译器优化能力非常强，这减轻了程序员的编程复杂度。
3) 使用预编译指令#pragma UNROLL(M)让编译器把内重循环全展开，这样编码的时候可以把1x8向量乘与8x8矩阵的操作写成循环，保持代码简洁灵活，在编译的时候编译器会自动做展开和优化。还有一个好处是，当编码参数M发生改变(例如不再是8，而改成其他值)的时候这部分代码不需要改变，只要修改M的宏定义就可以了。

```
void encoder_colPrio(
         IN  unsigned int  * restrict message, // Point to the message to be encoded
         OUT unsigned int  * restrict packets, // Point to the redundancy buffer
         IN  int blkSize, // The size of message block
         IN  int Rpackets) // The number of redundancy packets to be generated
{
    int i, col, row;
    long long msg_hilo, enc_hilo;
    unsigned int tmphi, tmplo;

    GPLYA = CRCPOLY;
    GPLYB = CRCPOLY;
    for (row=0; row < Rpackets ; row++)
    {
        for( i = 0; i < blkSize; i += 2 )
        {
            enc_hilo = 0;
            #pragma UNROLL(M) // M = 8
            for( col = 0; col < M; col++ )
            {
                msg_hilo = _amem8(&message[i + blkSize * col]);
                tmphi = _gmpy4(_hill(msg_hilo),crsOperator[row][col]);
                tmplo = _gmpy4(_loll(msg_hilo),crsOperator[row][col]);
                enc_hilo ^= _itoll(tmphi,tmplo);
            }
            _amem8(&packets[row*blkSize + i]) = enc_hilo;
        }
    }
}
```

使能 O3 优化后蓝色高亮显示的代码被编译成循环内核，需要 8 个 cycle 可以完成。M 单元受限。已经达到了 DSP 核处理能力的极限。软件流水线信息如下：

```
;*                                   A-side   B-side
;*       .L units                     0        0
;*       .S units                     0        0
;*       .D units                     4        6
;*       .M units                     8*       8*
;*       .X cross paths               2        3
;*       .T address paths             4        6
;*       Long read paths              0        0
;*       Long write paths             0        0
;*       Logical  ops (.LS)           3        5     (.L or .S unit)
;*       Addition ops (.LSD)          0        0     (.L or .S or .D unit)
;*       Bound(.L .S .LS)             2        3
;*       Bound(.L .S .D .LS .LSD)     3        4
;*
;*       Searching for software pipeline schedule at ...
;*          ii = 8  Schedule found with 4 iterations in parallel
```

除了循环内核的优化外，L1D cache 的访问性能也必须考虑。从代码结构可以看出计算每一行输出向量要同时用到 8 行输入向量。所以程序中要跳跃着读取数据包 M0,M1,..M7，每个包每次读取 8 Bytes。也就是有 8 个数据流。另一方面，C64，C64+和 C66 这些 DSP 核的 L1D cache 的大小是 32K Bytes，采用是 2-way 集相关结构。如果同时访问 2 路以上数据流并且每个数据流的指针跨度是 16K Bytes 的整数倍时，就会有一个以上的数据流被映射到 L1D 的同一个 cahe line，进而出现数据被反复刷入刷出 cache 的情况，导致 Cache 效率很低。所以必须避免这 8 路数据的读指针跨度是 16K Bytes 的整数倍。这对选择 CRS 编码数据包尺寸有一些限制。优化后的代码在 TI 6670 EVM 板上实测的 cycle 数如下：

表 1 TI 6670EVM 测试 cycle 表

Packet size L (Bytes)	6670 EVM cycle count	理论计算 cycle count L*R*M/8	6670 EVM cycle/ 理论计算 cycle （%）	冗余为 8 时（R=8） EVM 结果换算流量 Gbps @ 1Ghz
6144	56178	49152	114.29%	7.00
8192	216663	65536	330.60%	2.42
10240	93433	81920	114.05%	7.01
12288	111969	98304	113.90%	7.02
14336	129788	114688	113.17%	7.07
16384	520094	131072	396.80%	2.02
18432	166780	147456	113.10%	7.07
20480	186400	163840	113.77%	7.03
22528	205176	180224	113.84%	7.03
24576	649297	196608	330.25%	2.42
26624	242425	212992	113.82%	7.03
28672	260842	229376	113.72%	7.03
30720	277758	245760	113.02%	7.08

本文从理论上分析了计算量，验证了测试到的 cycle 是合理的：因为 CRS 编码函数的循环内核是 M 单元受限，所以我们以分析乘法计算量为主。做 M 个长度为 L 的数据包的 CRS 编码，生成 R 个校验包的乘法计算量是 L*M*R 次乘法。因为 DSP 核每个 cycle 可以做 8 次乘法。理论上需要的 cycle 数极限是 L*M*R/8。上表列出了理论极限和实测 cycle 数的差异，大部分场景有约有 15%的差异，这是 L1D cache miss 引入的延迟。当数据包尺寸 L 是 8192，16384， 24576 个 Bytes 的时候（也就是全部或部分读指针的跨度是 16K Bytes 的整数倍的时候），cycle 数突然升高。这印证了前面关于 L1D cache 访问特性的分析。从下图显示得更明显。

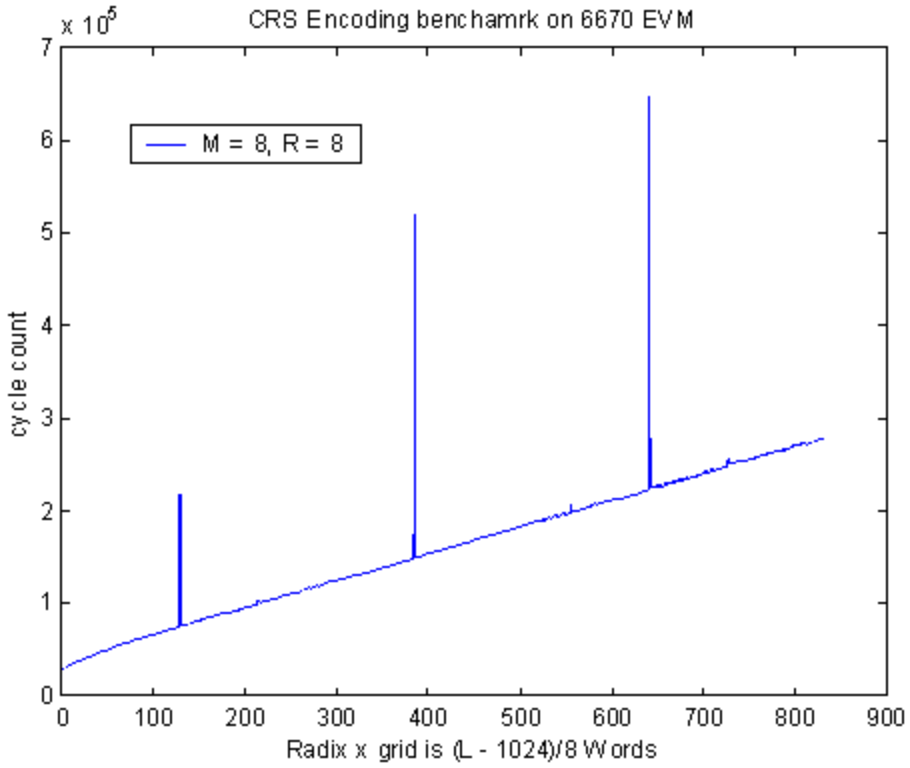

图 5 TI 6670 EVM 测试 cycle 示意图

5 总结

数据的安全性和可靠性在云时代对存储技术提出更高的要求.特别是随着硬盘的尺寸增大，传统的 RAID5 架构已经不能满足数据的可靠性要求。业界在切换到 RAID6 冗余时，也在寻找更高可靠性的方案，例如使用 RS 编码的方案来构建网络存储。

本文对 RS 算法做了详细的理论分析和算法描述，并在 TI 的 C6000 DSP 平台上评估了算法性能。TI C6000 DSP 支持专门的向量伽罗华域乘法指令，能够快速高效完成冗余信息的编码和数据恢复时的译码过程，单个 C6000 核能够提供超过 10Gbps 的 RAID6 加速（RAID6 的冗余备份 R=2），对于常用冗余为 3(R=3)的 CRS 编码，也可达到近于 10Gbps 的加速能力。TI 提供的部分 ARM SoC 芯片集成了 C6000 DSP 核，例如 66AK2E05 等，为支持数据冗余加速的网络存储应用提供灵活低成本解决方案。

参考文献

【1】 Johannes, An XOR-Based Erasure-Resilient Coding Scheme, 1995

【2】 James S. Plank, Optimizing Cauchy Reed-Solomon Codes, 2005

【3】 Wiki page at http://en.wikipedia.org/wiki/Invertible_matrix

重要声明

德州仪器(TI) 及其下属子公司有权根据 JESD46 最新标准, 对所提供的产品和服务进行更正、修改、增强、改进或其它更改， 并有权根据 JESD48 最新标准中止提供任何产品和服务。客户在下订单前应获取最新的相关信息, 并验证这些信息是否完整且是最新的。所有产品的销售都遵循在订单确认时所提供的TI 销售条款与条件。

TI 保证其所销售的组件的性能符合产品销售时 TI 半导体产品销售条件与条款的适用规范。仅在 TI 保证的范围内，且 TI 认为 有必要时才会使用测试或其它质量控制技术。除非适用法律做出了硬性规定，否则没有必要对每种组件的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。客户应对其使用 TI 组件的产品和应用自行负责。为尽量减小与客户产品和应 用相关的风险，客户应提供充分的设计与操作安全措施。

TI 不对任何 TI 专利权、版权、屏蔽作品权或其它与使用了 TI 组件或服务的组合设备、机器或流程相关的 TI 知识产权中授予 的直接或隐含权限作出任何保证或解释。TI 所发布的与第三方产品或服务有关的信息，不能构成从 TI 获得使用这些产品或服 务的许可、授权、或认可。使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可，或是 TI 的专利权或其它 知识产权方面的许可。

对于 TI 的产品手册或数据表中 TI 信息的重要部分，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况 下才允许进行复制。TI 对此类篡改过的文件不承担任何责任或义务。复制第三方的信息可能需要服从额外的限制条件。

在转售 TI 组件或服务时，如果对该组件或服务参数的陈述与 TI 标明的参数相比存在差异或虚假成分，则会失去相关 TI 组件 或服务的所有明示或暗示授权，且这是不正当的、欺诈性商业行为。TI 对任何此类虚假陈述均不承担任何责任或义务。

客户认可并同意，尽管任何应用相关信息或支持仍可能由 TI 提供，但他们将独力负责满足与其产品及在其应用中使用 TI 产品 相关的所有法律、法规和安全相关要求。客户声明并同意，他们具备制定与实施安全措施所需的全部专业技术和知识，可预见 故障的危险后果、监测故障及其后果、降低有可能造成人身伤害的故障的发生机率并采取适当的补救措施。客户将全额赔偿因 在此类安全关键应用中使用任何 TI 组件而对 TI 及其代理造成的任何损失。

在某些场合中，为了推进安全相关应用有可能对 TI 组件进行特别的促销。TI 的目标是利用此类组件帮助客户设计和创立其特 有的可满足适用的功能安全性标准和要求的终端产品解决方案。尽管如此，此类组件仍然服从这些条款。

TI 组件未获得用于 FDA Class III（或类似的生命攸关医疗设备）的授权许可，除非各方授权官员已经达成了专门管控此类使 用的特别协议。

只有那些 TI 特别注明属于军用等级或“增强型塑料”的 TI 组件才是设计或专门用于军事/航空应用或环境的。购买者认可并同 意，对并非指定面向军事或航空航天用途的 TI 组件进行军事或航空航天方面的应用，其风险由客户单独承担，并且由客户独 力负责满足与此类使用相关的所有法律和法规要求。

TI 已明确指定符合 ISO/TS16949 要求的产品，这些产品主要用于汽车。在任何情况下，因使用非指定产品而无法达到 ISO/TS16949 要求，TI不承担任何责任。

	产品		应用
数字音频	www.ti.com.cn/audio	通信与电信	www.ti.com.cn/telecom
放大器和线性器件	www.ti.com.cn/amplifiers	计算机及周边	www.ti.com.cn/computer
数据转换器	www.ti.com.cn/dataconverters	消费电子	www.ti.com/consumer-apps
DLP® 产品	www.dlp.com	能源	www.ti.com/energy
DSP - 数字信号处理器	www.ti.com.cn/dsp	工业应用	www.ti.com.cn/industrial
时钟和计时器	www.ti.com.cn/clockandtimers	医疗电子	www.ti.com.cn/medical
接口	www.ti.com.cn/interface	安防应用	www.ti.com.cn/security
逻辑	www.ti.com.cn/logic	汽车电子	www.ti.com.cn/automotive
电源管理	www.ti.com.cn/power	视频和影像	www.ti.com.cn/video
微控制器 (MCU)	www.ti.com.cn/microcontrollers		
RFID 系统	www.ti.com.cn/rfidsys		
OMAP应用处理器	www.ti.com/omap		
无线连通性	www.ti.com.cn/wirelessconnectivity	德州仪器在线技术支持社区	www.deyisupport.com

邮寄地址： 上海市浦东新区世纪大道 1568 号，中建大厦 32 楼 邮政编码： 200122