Turbo码的一种并行译码方案及相应的并行结构交织器研究

张曦林[①] 袁东风[①②]
[①](山东大学信息科学与工程学院 济南 250100)
[②](东南大学移动通信国家重点实验室 南京 210096)

摘 要 Turbo码基于MAP算法译码的递推计算所引入高的译码延迟限制了Turbo码在高速率数据传输中的应用。为了解决这个问题，该文提供了一种降低译码延迟的并行译码方法。并行处理方案的实现必须通过适当的交织以避免两个译码器对外信息读写的数据冲突。该文在分析了任意无冲突交织方式可能性的存在之后，给出了设计任意地适用于并行处理方案的S随机交织器的方法。仿真验证了并行译码方案的误比特性能。

关键词 Turbo码，译码延迟，并行处理，数据冲突，交织器
中图分类号：TN911.22 **文献标识码**：A **文章编号**：1009-5896(2006)06-1059-05

A Parallel Decoding Scheme and Parallel Construction Interleaver Turbo Codes

Zhang Xi-lin[①] Yuan Dong-feng[①②]
[①](*School of Information Science and Engineering, Shandong University*, *Jinan* 250100, *China*)
[②](*State Key Lab. on Mobile Communications, Southeast University*, *Nanjing* 210096, *China*)

Abstract The high latency introduced by the recursive computation in the MAP-based decoding of turbo codes limits the application of turbo codes in the high data rate transmission。In this paper, a parallel decoding scheme for reducing the decoding delay greatly is presented. In this parallel decoding scheme, interleaver (de-interleaver) must be collision-free when the extrinsic information memory is reading(or writing) between the two decoders. An idea of designing an arbitrary S-random interleaver using the collision-free mapping law is presented. Simulation results show that the BER performance of parallel decoding scheme is inferior slightly to that of the conventional scheme.

Key words Turbo codes, Decoding latency, Parallel decoding, Data collision, Interleaver

1 引言

Turbo码自 1993 年由Berrou[1]提出以来，由于其在低信噪比下所表现出来的接近香农限的性能，有关Turbo码设计及性能的理论研究成为国际信息与理论界的研究热点。同时Turbo码在各种通信系统中的应用与实现方法也引起了人们的极大兴趣。其中一个人们急于解决的问题是如何降低Turbo码的高的译码延迟，将其应用于高速数据通信中。

当Turbo码的接受序列很长，为了降低对硬件的存储要求并且为了降低译码延迟，通常采用滑动窗的方法[2]，即将接收到的序列分成W子段按照接收的顺序依次进行处理，对于每一子段，采用MAP算法或SOVA算法。但是，随着对处理速度要求的提高，需要采用比滑动窗方法更能提高数据吞吐量的方法，这就是本文研究的问题之一，即采用并行处理的方法，同时对W个子段进行译码。本文给出了Turbo码采用MAP算法并行译码处理的具体方案，为了不降低误比特性能，在一个子段的长度上交迭，计算虚拟前向递推$\alpha_k(m)$和虚拟后向递推$\beta_k(m)$，以合理地初始化每个子段的前向递推$\alpha_k(m)$和后向递推$\beta_k(m)$，并以此计算外信息$L_e(k)$。因为有W段的$L_e(k)$要参与交织，交织之后同时送入W个处理器进行译码，因此这种并行译码方案必须考虑两个译码器的SISO处理器和$L_e(k)$的存贮单元之间数据读写的冲突问题。而这个问题可以通过交织器的设计来解决。本文对免于数据冲突的交织(解交织)器进行了研究，分析了交织器设计的规则。并在这种规则下如何最大程度地提高交织器的随机扩展性进行了研究。计算机仿真的结果表明，并行译码的误比特性能相对于传统的译码方案略有差异，但每一个SISO译码处理器的运算量由传统的MAP译码方案时的$O(N)$降低为$O(N/W)$，译码延迟降低为原来的$1/W$。

2004-10-29 收到, 2005-05-15 改回
国家自然科学基金(60372030)，国家留学回国基金(教外司留[2003]406)和山东省自然科学基金重点项目(Z2003G02)资助课题

2 并行的基于MAP算法的译码结构

帧长为N的一帧数据被分割为W个子帧，每个子帧的数据长度为$WL=N/W$。与子帧数目相对应每个分量译码器

由 W 个 SISO 处理器组成，它们之间相互独立。

设对于第 1 个分量码译码器，第 w 个 SISO 处理器对子帧 w 由其前向递推状态度量值 α^w，后向递推状态度量值 β^w 计算外信息 L_{e1}^w。上标 w 表示子帧。

$$\alpha_k^w(m)=\frac{\sum_{m'}\sum_{i=0}^{1}\alpha_{k-1}^w(m')\cdot\gamma_i^w(R_k,m',m)}{\sum_{m}\sum_{m'}\sum_{i=0}^{1}\alpha_{k-1}^w(m')\cdot\gamma_i^w(R_k,m',m)},\quad w=1,\cdots,W,\ k=1,\cdots,WL \tag{1}$$

$$\beta_k^w(m)=\frac{\sum_{m'}\sum_{i=0}^{1}\beta_{k+1}^w(m')\cdot\gamma_i^w(R_k,m,m')}{\sum_{m}\sum_{m'}\sum_{i=0}^{1}\beta_{k+1}^w(m')\cdot\gamma_i^{w+1}(R_k,m,m')},\quad w=1,\cdots,W,\ k=WL-1,\cdots,1 \tag{2}$$

$$L_{e1,k}^w=\log\frac{\sum_{m}\sum_{m'}\gamma_1^w(y_k,m',m)\cdot\alpha_{k-1}^w(m')\cdot\beta_k^w(m)}{\sum_{m}\sum_{m'}\gamma_0^w(y_k,m',m)\cdot\alpha_{k-1}^w(m')\cdot\beta_k^w(m)},\quad w=1,\cdots,W,\ k=1,\cdots,WL \tag{3}$$

其中 $R_k=(x_k,y_k)$。

$$\gamma_i^w(R_k,m',m)=p(x_k\mid d_k=i)\cdot\gamma_i^w(y_k,m',m)\cdot\frac{\exp\{i\cdot L_{e2,k}^w\}}{\exp\{1+L_{e2,k}^w\}} \tag{4}$$

$$\gamma_i^w(y_k,m',m)=p(y_k\mid d_k=i,S_{k-1}=m',S_k=m)\cdot p(d_k=i\mid S_{k-1}=m',S_k=m) \tag{5}$$

在并行译码中，α^w，β^w 的初始化会对 BER 性能有所影响。为了得到 α^w，β^w 的初始值，可以在一个子帧的长度上交迭，做这样的运算：

$$\text{后向递推}\quad \alpha_0^1(m)=\begin{cases}1, & m=0,\\ 0, & m=1,\end{cases}\quad w=1$$

$$\alpha_0^w(m)=\alpha_{WL}^{w-1}(m),\quad w=2,\cdots,W \tag{6}$$

$$\text{后向递推}\quad \beta_{WL}^w(m)=\beta_1^{w+1}(m),\quad w=1,2,\cdots,W-1 \tag{7}$$

除第一个子帧以外，第 w 个子帧的前向递推初始值为上一个子帧 $w-1$ 的末尾时刻的值，在这里需要注意地是：对于第 w 个 SISO 处理器来说，第 $w-1$ 子帧上的前向递推的运算被称为虚拟前向递推运算。虚拟前向递推运算的起始值为

$$\alpha_0^{w-1}(m)=\frac{1}{2^M},\quad M\text{ 为编码器延迟单元的个数} \tag{8}$$

同此类似，第 w 个子帧的后向递推初始值 β_{WL}^w 为第 $w+1$ 子帧的开始时刻的值 β_1^{w+1}，对于第 w 个 SISO 处理器来说，第 $w+1$ 子帧上的后向递推的运算被称为虚拟后向递推运算。虚拟后向递推计算的起始值为

$$\beta_{WL}^{w+1}(m)=\alpha_{WL}^{w+1} \tag{9}$$

这里 α_{WL}^{w+1} 是有效运算的值。

虚拟计算的作用只在于得到前向递推和后向递推的初始值，而并不参与外信息的计算。与虚拟运算相对应的是有效运算，因此在得到初始值之后虚拟运算所得的值便再无价值，不必分配存储单元。虚拟运算是并行译码方案为了提高 BER 性能所付出的代价，在这里虚拟运算的长度取的是一个子帧，但是为了减少代价付出进一步减少译码延迟，可以使虚拟运算的长度减小，取编码器延迟单元个数 v 的 5－7 倍。在文献[3]中并行译码取虚拟运算长度 $N_{\text{dum}}=7v$。本文仿真结果比较了 N_{dum} 分别为 1 个子帧和 $7v$ 时并行译码的性能曲线。

外信息 $L_{e1,k}^w$ 的计算在有效后向递推 β_k^w 计算开始后进行，而并不需要等待所有的 β_k^w 算得之后。并行 Turbo 码译码器中的处理进程如图 1 所示。这是一个有 4 个 SISO 处理器，虚拟运算长度为一个子帧 WL 长的例子。在表示前项递推 α 和后向递推 β 的计算上，箭头的虚线部分表示虚拟运算，实线部分表示这个子帧的有效运算。

运算量的比较：如图 1 所示取虚拟长度 $N_{\text{dum}}=WL$，则前向递推 α 和后向递推 β 计算的总运算量为 $\Theta=3\times2WL+3\times2WL=12WL$；如图 2 虚拟长度若取 $N_{\text{dum}}=7v$，则 $\Theta=(4WL+3N_{\text{dum}})\times2=8WL+6N_{\text{dum}}$。

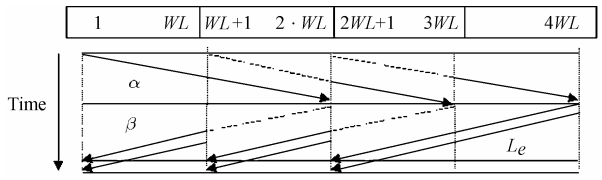

图 1 并行 Turbo 译码器中的处理过程

Fig.1 The parallel decoding process of Turbo codes

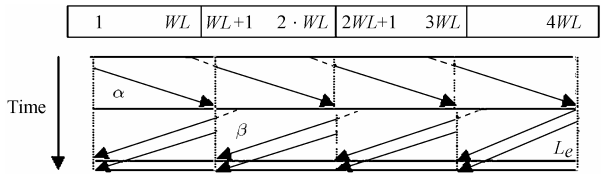

图 2 虚拟长度 N_{dum} 为 5～7 倍的并行处理

Fig.2 The parallel process when the dummy length $N_{\text{dum}}=5v\sim7v$

Turbo 码的这种并行处理的方法和滑动窗的处理方式很类似，不同的是滑动窗方法针对接收序列为半无限长(即 N 值很大)，并不需要将所有的一帧序列接收完毕，在译码结构上不需要有几个 SISO 处理器同时运算。如图 3 所示。

3 Turbo 码并行译码方案中的交织器

3.1 并行译码方案的交织映射规则

并行译码中的交织器如图 4 所示。

定义所有的变量为从 1 到 L，则第 j 个 SISO 处理从 $(j-1)\cdot WL+1$ 到 $j\cdot WL$ 的变量。与编码器 1 相对应的译码器

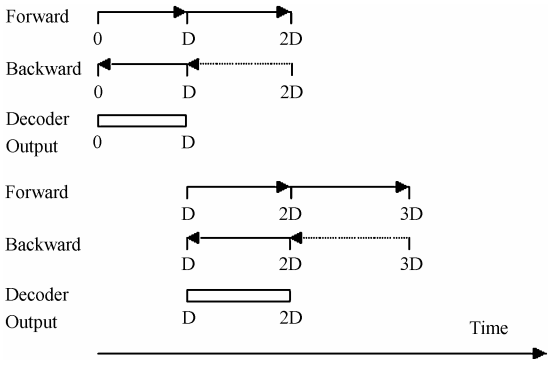

图 3　滑动窗的处理方式

Fig.3 The forward and backward processing for the sliding window

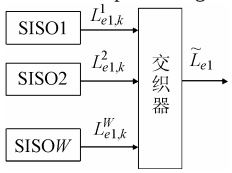

图 4 并行译码中的交织器

Fig.4 The interleaver in the parallel decoding process

1 的所图 4 有 SISOs 以自然顺序读写。在时刻 i，译码器 1 的 W 个 SISOs 读或写的变量为 $i,\ WL+i,\cdots,(W-1)\cdot WL+i$。与编码器 2 相对应的所有 SISOs，根据交织模式 π 来读写，在时刻 i，译码器 2 的 W 个 SISOs 读写的变量为 $\pi(i),\ \pi(WL+i),\cdots,\ \pi((W-1)\cdot WL+i)$。因此，并行问题可描述为给定一个存储数目为 W 的存储阵列，将译码器输出映射到存储阵列的映射方式为：在同一时刻 W 个输出(每一个由一个 SISO 产生)被映射到存储阵列中的不同的存储器，则不会发生数据冲突。定义一个函数对 $(M,S):\{1,\cdots,L\}\to\{1,\cdots,W\}\times\{1,\cdots,WL\}$ 具有以下的含义：对每一个译码器，第 i 个被写入存储阵列中标记为 $M(i)$ 的存储器，位置为 $S(i)$，则不产生数据冲突的并行限制条件表现为对 M 的限制：$\forall k,k'=1,\cdots,N,k\neq k'$，用 $=()_{WL}$ 表示对 WL 取模相等，有

$$k=(k')_{WL}\to M(k)\neq M(k') \tag{1}$$

$$k=(k')_{WL}\to M(\pi(k))\neq M(\pi(k')) \tag{2}$$

将 $\{1,\cdots,L\}$ 或 $\{\pi(1),\cdots,\pi(L)\}$ 写成 $W\times WL$ 的矩阵，则式(1)和式(2)的意义很明显：对译码器 1 和 2，无论以自然顺序或交织顺序，同一时刻从不同的存储器读出或写入。

文献[4]中给出一种方法，对任意给定的交织模式 π，找到一个 $\boldsymbol{M}:\{1,\cdots,L\}\to\{1,\cdots,W\}$ 的映射，然后根据 $\boldsymbol{M}$ 矩阵，用三层置乱的方式实现 π。这种方法对 (M,S) 中的 S 没有限制。这就证明了对于任意一种交织器，一定能保证无冲突的映射模式的存在。

把映射函数写成一个 $W\times WL$ 的矩阵 $\boldsymbol{M}$，矩阵中的元素 (i,j)，$i=1,\cdots,W,\ j=1,\cdots,WL$ 表示 $M((i-1)W+j)$，即 $(i-1)W+j$ 所在的存储器。

给定一个 k，定义下面两个集合：

$$C(k)=\{k':k'=(i-1)WL+k \bmod WL,i=1,\cdots,W\} \tag{3}$$

$$T(k)=\{\pi(k'):k'=(i-1)WL+k \bmod WL,i=1,\cdots,W\} \tag{4}$$

$C(k)$ 是 k 对 WL 取模的同余类，确定了 $\boldsymbol{M}$ 矩阵的列。$T(k)$ 为交织模式 π 在 $C(k)$ 上的值，确定了 $\boldsymbol{M}$ 矩阵的一个 Tile。$C(k)$ 和 $T(k)$ 定义了一个 $\boldsymbol{T}$ 矩阵。

这样式(1)就转化为对 $\boldsymbol{M}$ 矩阵的列的限制，式(2)依赖于交织模式 π，是对 $\boldsymbol{M}$ 矩阵中一个 Tile 的限制。

$$M(C(k))=\{1,\cdots,W\} \tag{5}$$

$$M(T(k))=\{1,\cdots,W\} \tag{6}$$

例如：给定一个交织 $\pi=(10,22,3,1,14,11,5,6,2,15,13,7,16,20,24,18,8,19,17,12,21,4,23,9)$，$L=24,\ W=4,\ WL=6$。

图 5 中的 $C(1)=\{1,7,13,19\}$ 定义了图 5(a)矩阵的第 1 列。$T(1)=\{10,5,16,17\}$ 定义了图 5(b)矩阵的第 1 列。

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

(a)

10	22	3	1	14	11
5	6	2	15	13	7
16	20	24	18	8	19
17	12	21	4	23	9

(b)

图 5　得到 $\boldsymbol{M}$ 矩阵的过程示例

Fig.5 The example of finding the $\boldsymbol{M}$ matrix

将 $T(1),T(2),T(3),T(4),T(5),T(6)$ 分别标记为 A，B，C，D，E，F，被称作 Tile，则得到图 5(b)矩阵的一个 $\boldsymbol{T}$ 矩阵如图 6(a)所示。

$\{1,\cdots,W\}$ 中的元素置入 $\boldsymbol{M}$ 矩阵的每一列和每一 Tile，每一列和每一 Tile 内不能有重复的元素。继续沿用上面的例子，例如得到的 $\boldsymbol{M}$ 矩阵如图 6(b)所示。

D	C	C	D	A	B
F	E	F	A	F	B
F	E	D	A	A	D
F	B	C	B	E	C

(a)

3	3	4	1	4	3
1	4	3	3	2	1
2	1	2	2	1	4
4	2	1	4	3	2

(b)

图 6　得到 $\boldsymbol{M}$ 矩阵的过程示例

Fig.6 The example of finding the $\boldsymbol{M}$ matrix

$\boldsymbol{M}$ 矩阵不可能一下就得到，因此可能有些位置保留空白，在这些空白位置填补元素，填补元素的过程称为 Annealing 过程。如图 7。

图 7(a)中的(1，2)和(4，1)位置为空白，Annealing 过程从一个空白开始。若从(1，2)开始，该列空缺的元素为 3。因此将 3 填补在该处；但是(1，2)和(1，3)属于同一个 Tile C，它们的值都为 3，引起了冲突。所以将(1，3)内的值改变，填入该 Tile 内还没有被分配的值 4，如图 7(b)所示。

这时在第 3 列内有两个 4 存在，因此将(2，3)改为 3，改变后在(2，3)所在的 Tile 内没有引起冲突。

检查现在所得到的 $\boldsymbol{M}$ 矩阵，在(1，4)处为空白，因此，填入 4，没有引起冲突，如图 7(c)所示。这样，就完成了整个 Annealing 过程。

3		3	1	4	3
1	4	4	3	2	1
2	1	2	2	1	4
	2	1	4	3	2

(a)

3	3	4	1	4	3
1	4	4	3	2	1
2	1	2	2	1	4
	2	1	4	3	2

(b)

3	3	4	1	4	3
1	4	3	2	3	1
2	1	2	5	1	4
4	2	1	4	2	2

(c)

图 7 得到 $\boldsymbol{M}$ 矩阵的过程示例

Fig.7 The example of finding the $\boldsymbol{M}$ matrix

总结上面的 Annealing 过程，可以分为几轮，每轮从一个空白处开始，当不再有冲突产生的时候结束；然后再从一个空白处开始另一轮。在一轮内的 Annealing 过程总结如下：

定义 $L(C(k))$ (同样地有 $L(T(k))$)为 $1,\cdots,W$ 中没有分配给 $C(k)$ ($T(k)$)的值的集合。

在一个空白位置 $k^{(0)}$，假如 $L(C(k))$ 和 $L(T(k))$ 的交集为非空，则存在一个值可以填补 $k^{(0)}$ 的空白而不会引起冲突，这时找到 $M(k^{(0)})$ 。

假若 $L(C(k))$ 和 $L(T(k))$ 的交集为空集，选择两个值，$m\in L(C(k^{(0)}))$ 和 $n\in L(T(k^{(0)}))$，则指定 $M(k^{(0)})=m$ 并且重复以下过程：

(1) 找到 $k^{(1)}\in T(k^{(0)})$，有 $M(k^{(1)})=m$ 并且 $k^{(1)}\neq k^{(0)}$ 。如果找到了这样的 $k^{(1)}$，令 $M(k^{(1)})=n$，若在 $C(k^{(1)})$ 内不引起冲突，则该轮结束；否则继续下一步。

(2) 找到 $k^{(2)}\in C(k^{(1)})$ 有 $M(k^{(2)})=n$ 并且 $k^{(2)}\neq k^{(1)}$，如果找到这样的 $k^{(2)}$，就分配 $M(k^{(2)})=m$，若在 $T(k^{(2)})$ 内不引起冲突则该轮结束。否则，找 $k^{(2i-1)}\in T(k^{(2i-2)}),i=2,\cdots$，重复(1)和(2)的过程。

一般来说对一个交织模式 π，总能找到与之对应的 $\boldsymbol{M}$ 矩阵。

3.2 交织器的实现

找到映射矩阵 $\boldsymbol{M}$ 后，交织过程分为 3 层置乱实现：

第 1 层 一个时变的置乱函数，有 W 个输入，W 个输出，将 $\{1,\cdots,L\}$ 分时刻 $j=1,\cdots,WL$ 映射到各自的 $M(\cdot)$ 。

第 2 层 由 W 个交织器构成，交织长度为 WL，每个交织器标记为 π_i，它是将 $M(k)=i$ 的 k 进行交织，交织规则为 $j=(k)_{WL}, j'=(\pi^{-1}(k))_{WL}$， $\pi_i(j')=j$ 。

第 3 层 一个时变的置乱函数，有 W 个输入和 W 个输出，对于 j 时刻从 W 个交织器的输入 $i=1,\cdots,W$，有输出 $M(\pi(i-1)\cdot WL+j)$ 。

例如 在 $j=1$ 时刻，第 1 层将 1，7，13，19 分别映射到

$$M(1)=3, M(7)=1, M(13)=2, M(19)=4$$

$M(k)=1$ 的 k 有 4，7，12，14，17，21；由 $j=(k)_{WL}$，得 j =4，1，6，2，5，3。

第 2 层的交织器 π_1 将使 $M(k)=1$ 的 k 按照读出的时间顺序交织。由于 $\pi^{-1}(4)=22,\ \pi^{-1}(7)=12, \pi^{-1}(12)=20, \pi^{-1}(14)=5, \pi^{-1}(17)=19, \pi^{-1}(21)=21$，则 $j'=(\pi^{-1}(k))_{WL}$ 分别为 4，6，2，1，3， 交织的规则为 $\pi_1(j')=j$ 。相应的 π_2,π_3,π_4 进行各自的交织。

第 3 层在 $j=1$ 时刻，$i=1,2,3,4$ 个输出分别指向 $M(\pi(1)=10)=3$，$M(\pi(7)=5)=4$，$M(\pi(13)=16)=2$，$M(\pi(19)=17)=1$。

4 利用并行映射规则设计 S 随机交织器

文献[4]给出了一种一般性的方法，说明任何一种交织模式经过 3 层置乱过程，都可以达到并行运算的目的。先给定一个交织模式 π，找到相应的映射矩阵 $\boldsymbol{M}$，然后经 3 层置乱以给出交织后的数据。找到 $\boldsymbol{M}$ 矩阵，以及 3 层置乱的实现都比较复杂。如果我们是要找到对于每一个 SISO 扩展性好的 S 随机交织模式，只要经过一层时变的置乱过程即可以获得随机性强的交织模式，使之适用于并行计算。下面给出一种形成 S 随机交织器的实现方法：

在上一节中，映射规则由 $\{1,\cdots,L\}\to\{1,\cdots,W\}\times(1,\cdots,WL)$ 只对 $M(k)$ 进行了限制，而对 $S(k)$ 并没有限制，在这里做出改变，即设计 S 随机交织器时对 k 在 $M(k)$ 中的位置 $S(k)$ 也加以限制。

(1) 在 $j=1$ 时刻，将 $k=1,1+W,\cdots,1+(i-1)WL,\cdots,1+(W-1)\cdot WL$ 随机选择 $M(k)$，$M(k)\in\{1,\cdots,W\}$。$S(k)=1$。

(2) 在 $j>1$ 时刻，将 $k=j, j+WL,\cdots, j+(W-1)\cdot WL\in\{1,\cdots,L\}$ 的 W 个并行输出分别被映射到 $M(j), M(j+WL),\cdots, M(j+(W-1)\cdot WL)$，$k$ 在 $M(k)$ 内的位置为 $s(k)=j$。由 $k\to M(k)$ 的规则为 k 与 $M(k)$ 中的其它元素满足 S 随机交织器的限制条件：$S\le\sqrt{L/2}$。具体描述如下：(a) 取 $M(j)=1$，若 j 与 $M(j)$ 内的其它元素不满足 S 随机交织的限制，则将 $M(j)=M(j)+1$，直到找到满足 S 随机交织限制的 $M(j)$；(b) 查找 $M(j+(i-1)\cdot WL)\in\{1,\cdots,W\}$)但 $\neq M(j),\cdots,M(j+(i-2)\cdot WL)$ 使之满足 S 随机交织限制，若找不到合适的 $M(j+(i-1)\cdot WL)$，则回到(a)重新安排 $M(j)$。直到所有的 k 都找到满足 S 随机交织条件的 $M(k)$。

(3) $j=j+1$，重复(2)的过程。

例如：$S=3$ 的 S 随机交织器如图 8 所示。

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

→

1	8	21	4	11	24
7	14	3	10	17	6
19	2	15	22	5	18
13	20	9	16	23	12

图 8 $S=3$ 的 S 随机交织器的例子

Fig.8 The example of S-random interleaver for $S=3$

5 仿真结果

本文采用 4SISO处理器基于MAP算法对 1024 bit的帧长进行了仿真，在AWGN信道下，采用S随机交织器，生成矩阵采用(1，13/15)。仿真结果如图 9 所示。4SISO处理器 $N_{\text{dum}}=256$ bit与 1SISO处理器进行比较的结果，可以看出，

两者在 0.1 dB －0.4dB 低信噪比的时候性能上基本上没有差异的；在 0.4dB-0.7dB期间，1 个SISO处理器组成的译码器译码的BER性能，要比 4 个SISO处理器构成的译码器译码的下降速度快，在 0.7dB时，都能达到 10^{-5}的BER性能。由于在0.7dB以上两者的误码率都很低，计算机仿真需要的数据量很大，可能会造成比较大的误差，因此可以用联合限的办法来估计在高信噪比的条件下两者的性能差异，这也是作者以后深入研究的一个方向。另外，同样为 4SISOs，但 N_{dum} 不同时BER性能也有差异，随着 N_{dum} 长度的减小，BER性能会略有下降。

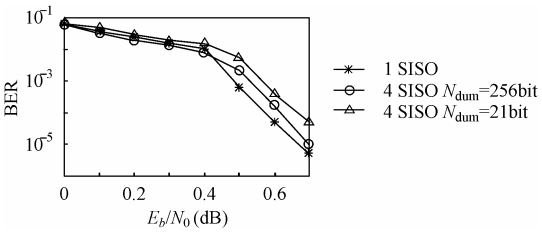

图 9　1SISO 与 4SISO 处理器以及 N_{dum} 不同时的仿真结果比较

Fig.9　Comparison between 1SISO and 4SISO processor and between different N_{dum} lengths

6　结束语

Turbo 应用于各种通信系统，在利用它的在低信噪比情况下接近香农限的优异性能的情况下，有时还必须考虑如何降低 Turbo 码的译码延迟。本文针对 Turbo 码基于 MAP 算法的降低译码延迟的并行译码方案进行深入研究，并且对于这种并行译码方案硬件实现中两个译码器之间的免于数据冲突的交织器问题进行了研究，给出了免于数据冲突的 S 随机交织器设计方法。从仿真的结果来看，尽管并行译码方案在性能上略微有所下降，但译码延迟和每个 SISO 处理器的运算量以及存储规模大大降低，更适宜于在实际通信系统中帧长较长，而且实时性和数据速率要求高的情况。

参 考 文 献

[1] Berrou C, Glavieux A, Thitimajshima P. Near Shannon limit error-correcting coding and decoding: Turbo-codes(1). in Proc.ICC'93, Geneva, May 1993: 1064−1070.

[2] Benedetto S, Montorsi G, Divsalar D, Pollara F. A soft-input soft-output maximum a posterior (MAP) module to decode parallel and serial concatenated codes. JPL TDA Progress Report, 42-127, November 1996.

[3] Jaeyoung Kwak, Kwyro Lee. Design of dividable interleaver for parallel decoding in turbo codes. *Electronics Letters*, 2002, 38(22): 1362−1364.

[4] Tarable A, Benedetto S. Mapping interleaving laws to parallel turbo decoder architectures. *IEEE Communications Letters*, 2004, 8(3): 162−164.

张曦林：　女，1974 年生，硕士，研究方向为移动通信中的关键技术、信道编码等.

袁东风：　男，1958 年生，教授，博士生导师，现从事移动通信中纠错抗干扰技术的研究.