第34卷 第14期 计 算 机 工 程 2008年7月
Vol.34 No.14 Computer Engineering July 2008
·博士论文· 文章编号：1000—3428(2008)14—0009—03 文献标识码：A 中图分类号：TP301.6

一种用于规则 QC-LDPC 码的高效译码方法

赵 岭[1]，张晓林[1]，朱曼洁[2]
(1. 北京航空航天大学电子信息工程学院，北京 100083；2. 中国空间技术研究院卫星应用系统部，北京 100083)

摘 要：针对一类规则 QC-LDPC 码，提出一种高效的 Log-BP 译码方法，通过矩阵分裂，将原监督矩阵分裂成多个小矩阵，将原本的校验节点更新运算拆分成多次处理，以降低 log-BP 迭代运算的复杂度，给出该方法的迭代运算顺序。与现有的 log-BP 译码方法相比，该方法在相同的码速率下，校验节点运算单元与变量节点运算单元总规模减小了 1/3；在相同的硬件资源下，译码速率提高了 1/3，且校验节点运算单元与变量节点运算单元结构趋于对称，有利于设置更少的流水线级数、获得更好的时钟性能。

关键词：QC-LDPC 码；log-BP 算法；矩阵分裂

Efficient Decoding Method for Regular QC-LDPC Codes

ZHAO Ling[1], ZHANG Xiao-lin[1], ZHU Man-jie[2]
(1. School of Electronics and Information Engeering, Beijing Univ. of Aeronautics and Astronautics, Beijing 100083;
2. Satellite Application System Department, China Academy of Space Technology, Beijing 100083)

【Abstract】 This paper presents a decoding method for regular QC-LDPC codes. In this method, the original check matrix is split into several smaller ones, thus CNU is decomposed into multiple parallel units, which results in great hardware resource reduction. A novel iterative sequence is presented. Compared with the traditional log-BP decode method, the method can reduce the whole logic core size of CNU and VNU by 1/3 under the same bit-rate, and it can improve bit-rate by 1/3 under the same logic core size in the condition of the original check matrix split into two parts.

【Key words】 QC-LDPC codes; log-BP algorithm; matrix split

1 概述

低密度奇偶校验(Low Density Parity Check, LDPC)码是一类可用非常稀疏的奇偶校验(parity-check)矩阵或二分图(bipartite graph)定义的线性分组纠错码，最早由文献[1]提出。LDPC 码采用迭代译码算法，性能优异，接近于香农限，并被证明与 Turbo 码的性能相当甚至更优，因此，成为近年来纠错码领域的研究热点，并广泛应用于高可靠性通信和数字存储等系统中。但是解码难度大大小于后者。LDPC 码已成功应用于欧洲数字电视卫星传输标准 DVB-S2 中。我国的数字电视地面广播传输标准也采用 LDPC 码作为信道编码。

置信概率传播(BP)算法通常用来对 LDPC 进行译码，在 BP 译码算法中，有 2 种信息根据 Tanner 图传递：(1)通过变量节点运算单元(VNU)传递的变量节点到校验节点的信息；(2)校验节点运算单元(CNU)传递的校验节点到变量节点的信息。就具体实现而言，部分并行结构比全并行结构简单，更具有实用价值。

文献[2]提出了一种改进的译码结构，将变量节点运算单元与校验节点运算单元复用，大大地节省了硬件资源。但是，只有当 LDPC 码的行重与列重相差不大时，该方法才有明显的优势，而对于高码率的 LDPC 码，例如 CCSDS 标准中推荐的 LDPC 码(行重为 32，列重仅为 4)，则不太适用。

2 Log-BP 译码算法

兼顾译码性能与复杂度 2 方面，本文采用log-BP算法[3]进行译码，其中，译码器的输入为来自信道的先验概率 $p_j^0 = p(x_j = 0)$, $j \in (1\sim N)$，输出为译码硬判决结果。迭代过程中的中间变量为$L(q_{mj})$和R_{mj}。其中，m表示校验节点；j表示变量节点；$L(q_{mj})$为变量节点到校验节点的迭代信息；R_{mj}为校验节点到变量节点的迭代信息。令$M(j)$表示与比特节点j相连的校验节点的集合；$N(m)$表示与校验节点m相连的变量节点的集合，具体步骤如下：

(1)初始化。对所有 $j \in (1\sim N)$, $m \in M(j)$，计算

$$L(q_{mj}) = \ln\left(p_j^0 / p_j^1\right) = -2r_j / \sigma^2 \tag{1}$$

(2)迭代：

1)校验节点更新运算(Check Node Update Processing, CNUP)由 CNU 完成，对所有 m, n，计算

$$R_{mj} = \prod_{\substack{n\in N(m)\\ n\neq j}} \operatorname{sign}\left(L(q_{mn})\right) \times \Psi\left(\sum_{n\in N(m), n\neq j} \Psi\left(L(q_{mn})\right)\right) \tag{2}$$

其中，$\Psi(x) = \ln\left(\tanh\left(|x/2|\right)\right) = \ln\left(\left(1-\mathrm{e}^{-|x|}\right)/\left(1+\mathrm{e}^{-|x|}\right)\right)$。

2)变量节点更新运算(Variable Node Update Processing, VNUP)由 VNU 完成，对所有 m, n，计算

$$L(q_j) = \sum_{m\in M(j)} R_{mj} + \left(-2r/\sigma^2\right) \tag{3}$$

$$L(q_{mj}) = L(q_j) - R_{mj} \tag{4}$$

(3)对$L(q_j)$判决。对所有$j \in (1\sim N)$，若$L(q_j) \geqslant 0$，则$x_j=0$，否则，$x_j=1$。若$\boldsymbol{H}\cdot\boldsymbol{x}=0$，译码正确；否则继续迭代，直到最大迭代次数。

3 QC-LDPC 码及部分并行译码结构

QC-LDPC码[4]是近几年研究得最多的一类LDPC码，它的校验矩阵可由若干循环行列式描述。其校验矩阵的结构表示如下：

作者简介：赵 岭(1980－)，男，博士研究生，主研方向：信道编解码技术；张晓林，教授、博士生导师；朱曼洁，工程师、硕士

收稿日期：2007-08-30 **E-mail**：zhaoling@ee.buaa.edu.cn

$$\boldsymbol{H}=\begin{bmatrix} \boldsymbol{A}_{0,0} & \boldsymbol{A}_{0,1} & \cdots & \boldsymbol{A}_{0,p} \\ \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} & \cdots & \boldsymbol{A}_{1,p} \\ \vdots & \vdots & & \vdots \\ \boldsymbol{A}_{q,0} & \boldsymbol{A}_{q,1} & \cdots & \boldsymbol{A}_{q,p} \end{bmatrix} \tag{5}$$

其中，$\boldsymbol{A}_{i,j}$是一个$m\times m$的 0 矩阵或是由循环行列式构成的方阵，且循环行列式的重量很小。如果$\boldsymbol{H}$矩阵每一行的重量相等并且每一列的重量也相等，那么$\boldsymbol{H}$对应一个规则的LDPC码，否则对应一个非规则的LDPC码。

根据 QC-LDPC 码校验矩阵的结构特点，可以采用部分并行译码结构对 QC-LDPC 码使用 log-BP 译码算法，即各行块间的校验节点更新运算与各列块间的变量节点更新运算是并行执行的，而各行块内的校验节点更新运算与各列块内的变量节点更新运算是是串行执行的。还可以在部分并行结构中取不同的并行度。

例如，对式(5)所示的校验矩阵对应的LDPC码进行译码，可以把行并行度设成q+1，即有q+1 个CNU运算单元：CNU_0, CNU_1,…, CNU_q，分别完成$\boldsymbol{A}_{0,j}$~$\boldsymbol{A}_{q,j}$ (j=0,1,…,p)的校验节点更新运算；将列并行度设成p+1，那么共有p+1 个VNU运算单元：VNU_0, VNU_1,…,VNU_p，分别完成$\boldsymbol{A}_{i,0}$~$\boldsymbol{A}_{i,p}$ (i=0,1,…,q)的变量节点更新运算。所以，要完成整个矩阵的校验节点更新运算，需要q+1 个CNU运算单元并行工作m个时钟周期；要完成整个矩阵的变量节点更新运算，需要p+1 个VNU运算单元并行工作m个时钟周期，即完成一次迭代需要 2m个时钟周期。也可以把行并行度设成(q+1)/2 (为方便叙述，假设q+1能被 2 整除，这不影响结果)，列并行度不变，那么，CNU的数目变成了(q+1)/2，VNU的数目还是p+1，此时，需要(q+1)/2个CNU运算单元首先并行工作m个时钟周期，完成校验矩阵前(q+1)/2 个行块的校验节点更新运算，然后，这些CNU运算单元继续并行工作m个时钟周期，完成校验矩阵后(q+1)/2 个行块的校验节点更新运算，所以，共需要 2m个时钟周期才能完成整个矩阵的校验节点更新运算。另一方面，由于列的并行度不变，因此完成变量节点更新运算的时间不变。此时，完成一次译码的时间需要 3m个时钟周期，整个译码的运算顺序如图 1 所示。

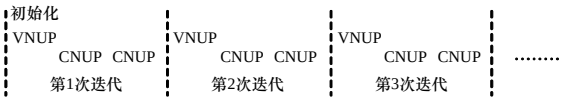

图 1　部分并行结构下的迭代顺序

当然，也可以降低列的并行度，或同时降低行和列的并行度。但随着行、列并行度的降低，译码器的译码速率会降低。表 1 列举了在不同的行列并行度下，CNU 总规模、VNU 总规模以及译码速率的对比情况。

表 1　不同并行度下的性能对比

方法	行并行度	列并行度	CNU 总规模	VNU 总规模	译码速率
1	q+1	p+1	1	1	1
2	(q+1)/2	p+1	1/2	1	2/3
3	q+1	(p+1)/2	1	1/2	2/3
4	(q+1)/2	(p+1)/2	1/2	1/2	1/2

在表 1 中，CNU 总规模、VNU 总规模以及译码速率均以方法 1 的结果归一化。可以看出，如果把行并行度降一半(方法 2)或者将列并行度降一半(方法 3)，那么 CNU 以及 VNU 总规模的和将减少 1/4(CNU 总规模与 VNU 总规模相当，均设为 1)，但是译码速率降低了 1/3；如果把行并行度和列并行度都降一半(方法 4)，那么 CNU 以及 VNU 总规模的和将减少 1/2，而译码速率降低了 1/2。由此可见，对于传统的译码结构，如果减少硬件运算单元的规模，译码速率会大大降低。本文提出了一种新的译码结构，在减少硬件运算单元的规模的同时，付出的译码速率的代价较传统的方法小得多。

4　基于矩阵分裂的高效译码方法

以(4, 6)规则QC-LDPC码为例。首先对它的奇偶校验矩阵进行分裂，分裂方法如图 2 所示，$\boldsymbol{H}$为原始矩阵，分别将它的前半列块与后半列块构成 2 个新的矩阵$\boldsymbol{H}_0$和$\boldsymbol{H}_1$。

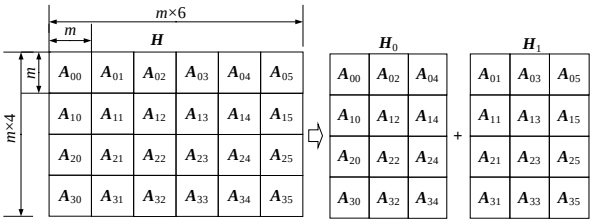

图 2　校验矩阵分裂示意图

显然，$\boldsymbol{H}_0$和$\boldsymbol{H}_1$的变量节点更新运算与$\boldsymbol{H}$的变量节点更新运算相对应，但是，原矩阵$\boldsymbol{H}$的校验节点更新运算需要$\boldsymbol{H}_0$和$\boldsymbol{H}_1$相互配合才能完成。矩阵分裂后的部分并行译码结构如图 3 所示，其中包含了 4 个 3 输入的CNU、3 个 4 输入的VNU、24 个用于存储中间迭代变量迭代存储器m_{mj}、4 个用于存储$\boldsymbol{H}_0$和$\boldsymbol{H}_1$校验节点运算单元交互信息的存储器m_m。

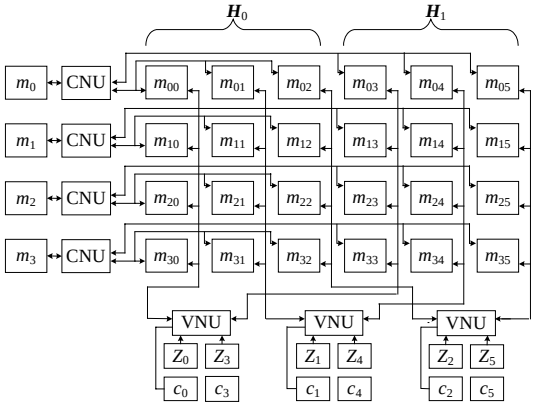

图 3　基于矩阵分裂的部分并行译码结构

原矩阵$\boldsymbol{H}$分裂成$\boldsymbol{H}_0$和$\boldsymbol{H}_1$后，需要$\boldsymbol{H}_0$的CNU单元运算 2 次、$\boldsymbol{H}_1$的CNU运算单元运算 1 次才能完成$\boldsymbol{H}$校验节点信息的更新，同样，需要$\boldsymbol{H}_0$和$\boldsymbol{H}_1$的VNU运算单元各运算一次才能将$\boldsymbol{H}$的变量节点信息更新，如果完全串行，则需要 5m个时钟周期才能完成一次迭代。但由于$\boldsymbol{H}_0$和$\boldsymbol{H}_1$具有相对独立性，因此可以将这些运算并行起来，运算顺序如图 4 所示，其中，CNUP_0和VNUP_0分别表示$\boldsymbol{H}_0$的校验节点更新运算与变量节点更新运算，而CNUP_1和VNUP_1分别表示$\boldsymbol{H}_1$的校验节点更新运算与变量节点更新运算。

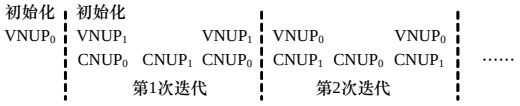

图 4　矩阵分裂后的迭代顺序

如图 4 所示，对$\boldsymbol{H}_0$初始化后，可同时进行$\boldsymbol{H}_1$的初始化以及$\boldsymbol{H}_0$的校验节点更新的第 1 次运算，得到一个中间结果；接着进行$\boldsymbol{H}_1$的第 1 次校验节点更新运算，算出$\boldsymbol{H}$的奇数列块的校验节点更新结果；然后将$\boldsymbol{H}_1$的变量节点更新运算和$\boldsymbol{H}_0$的第

2 次校验节点更新运算并行起来。到此，第 1 次迭代的校验节点更新运算完成，后面的运算类似。因此，用本文的译码方法完成一次迭代的时间是 3m个时钟周期。

由于$\boldsymbol{H}_0$或者$\boldsymbol{H}_1$的规模只有$\boldsymbol{H}$的一半，因此$\boldsymbol{H}_0$, $\boldsymbol{H}_1$的CNU规模和VNU的规模只有$\boldsymbol{H}$的一半。

由以上算法还可以看出，在矩阵分裂前后，VNU 的实现结构相同，但是 CNU 略微有些区别，新的 CNU 比原 CNU 多了一个中间变量输入以及一个中间变量输出，它对应的实现结构如图 5 所示。

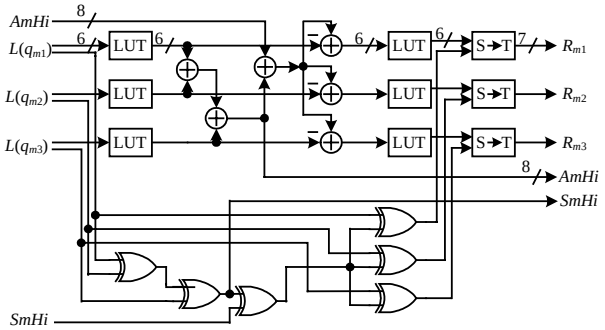

图 5　矩阵分裂后的 CNU 运算单元结构

本文方法的各项指标总结如下:

行并行度	列并行度	CNU 总规模	VNU 总规模	译码速率
q+1	(p+1)/2	1/2	1/2	2/3

与表 1 对照可以看出，本文方法与方法 2、方法 3 的译码速率相等，均为 2/3，但是本文方法的 CNU 与 VNU 总规模比方法 2、方法 3 减小了 1/3；同时，与方法 4 的 CNU 与 VNU 总规模相当，均为 1，但是码速率比方法 4 快了 1/3。另外,本文方法减小了单个 CNU 运算单元的规模,使得 VNU 与 CNU 的复杂度趋于平衡,通过增加较少的流水线可以使译码器在更高的频率下工作。

5　结束语

本文提出了一种新的基于矩阵分裂的规则 QC-LDPC 的部分并行译码方法，它能有效降低实现的代价，提高译码速率，特别适用于高码率、大长度的 QC-LDPC 码。本文仅仅给出了矩阵分裂度为 2 时的译码方法，对于码长更长的高码率 QC-LDPC 码，可以把分裂度设成 3 或者更大。可以证明，若分裂度为 n,与传统方法相比,本文方法在译码速率或 CNU 和 VNU 运算总规模上的优势是 1/(2n-1)。

参考文献

[1] Gallager R G. Low Density Parity Check Codes[J]. IRE Trans. on Information Theory, 1962, 8(1): 21-28.

[2] Wang Zhongfeng, Chen Yanni, Parhi K K. An Aera Efficient Decoding of Quasi-cyclic Low Density Parity Check Codes[C]// Proc. of ICASSP'04. [S. l.]: IEEE Press, 2004.

[3] Zhang T, Parhi K K. VLSI Implementation Oriented (3,k)-regular Low-density Parity-check Codes[C]//Proc. of SIPS'01. [S. l.]: IEEE Press, 2001.

[4] Sridhara D, Fuja T, Tanner R M. Low Density Parity Check Codes from Permutation Matrices[C]//Proc. of Conf. on Info. Sciences and Systems. [S. l.]: John Hopkins University Press, 2001.

(上接第 8 页)

5　相关工作与讨论

Sun设计了一种支持网格应用的分布式虚拟机[3],为了支持动态的资源管理，系统引入了 2 种基本资源管理抽象，都采用Java的库函数实现。Isolate代表一个任务，是资源管理的基本单位。Resource domain代表一个物理资源与资源管理策略。Grid Job Engine将根据可用资源和资源管理策略进行资源管理。和AVM不同，Sun的分布式虚拟机需要在程序中显示指定任务所需资源和资源管理策略。而AVM则是在运行时由环境自动完成资源管理。另外，在Sun的分布式虚拟机中的资源管理单元中，一个isolate实际上是一个单独运行的Java程序；而AVM则是以服务为单位进行资源管理的。

Coign[4]是微软提出来的基于组件的自动分布系统。和本文的工作类似,Coign希望能自动将应用程序分布在多台主机上执行。然而，Coign使用了一种静态的分析方式试图找出通信量最小的分布方案。由于这是一个NP问题，因此Coign只实现了 2 台主机的自动分布,它是一种静态的资源分布方案,并不适用于网格环境中资源动态变化的需求。

此外，一些网格应用程序使用部署语言与运行环境隐式部署[5]，例如glassfish利用Java1.5 中的annotation特性，允许在程序中隐式指定部署规则。以上这些相关工作在一定程度上提高了资源管的的自动化程度，但实际上都是一种静态的部署方式，并没有考虑到实际运行环境的资源情况。

6　结束语

为解决现有网格运行环境中静态资源管理带来的问题，本文基于一种新的动态资源管理模式，设计并实现了一种基于分布式虚拟机的网格运行环境 AVM。根据面向服务的网格应用的特点,AVM主要以3种技术来优化动态资源管理系统:(1)借助面向服务的开发语言，以服务作为系统基础抽象；(2)利用服务的无状态性，采用按需部署技术实现服务的远程部署；(3)采用服务部署成本分析模型，根据应用的资源管理策略和当前可用资源决定服务部署方案。测试结果显示，这种资源管理技术取得了预计效果。下一步将研究如何完善成本分析模型以支持其他资源管理策略，以及如何利用服务的无状态特性，隐式地提高应用程序的并发程度。

参考文献

[1] Wang Xiaoning, Xiao Lijuan, Li Wei, et al Abacus: A Service-oriented Programming Language for Grid Applications[C]//Proc. of IEEE International Conference on Services Computing. Orlando, Florida, USA: [s. n.], 2005.

[2] 龚奕利, 李　伟, 孙毓忠. 网格环境中资源发现方法研究[J]. 计算机工程, 2006, 32(17): 162-164.

[3] Wegiel M, Czajkowski G, Dqynes L, et al. A Portable Grid Infrastructure for Resource-aware Applications[C]//Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid. Singapore: [s. n.], 2006.

[4] Hunt G, Scott M L. The Coign Automatic Distributed Partitioning System[C]//Proc. of the 3rd Symposium on Operating Systems Design and Implementation. New Orleans, LA, USA: [s. n.], 1999.

[5] Glassfish Project[Z]. (2007-03-01). http://java.sun.com/javaee/community/glassfish.