Redução de profundidade de lógica de hardware para codificação de LDPC

Alexandre Felipe e André L.N. Souza

***Resumo*— As matrizes de paridade esparsas de códigos LDPC que permitem codificação com complexidade linear normalmente geram implementações em *hardware* com grande profundidade de lógica. O objetivo desse trabalho é indicar maneiras de manipular um sistema de equações triangular para diminuir o número de estágios não paralelizáveis, reduzindo assim a latência do sistema de codificação e permitindo a transmissão de taxas mais elevadas.**

***Palavras-Chave*— Códigos com matriz de paridade esparsa, LDPC, Latência, Codificação.**

***Abstract*— Low-density Parity-check codes constructed for coding with linear computational complexity normally generate encoder hardware with high logical depth. This work indicates strategies to manipulate a triangular equation system in order to reduce the non-parallelizable stages, and consequently lower enconding latency, allowing faster bit rates.**

***Keywords*— Low-density parity-check codes, LDPC, Latency, Encoding.**

I. Introdução

A existência de códigos LDPCs que são capazes de alcançar assintoticamente com o tamanho da palavra código o limite da capacidade de correção de erros é conhecida desde a década de sessenta [3]. Com o aumento da capacidade computacional, na década de noventa houve um grande avanço nas técnicas de correção de erros, em uma época marcada pela descoberta dos Códigos Turbo [1]. Quatro anos depois, a viabilidade dos códigos LDPCs foram demonstrados por Mackay et al. [5]. Em 2001 demonstrou-se que os códigos LDPCs poderiam ter desempenho muito melhor que os Códigos Turbo [7]. Desde então os códigos LDPC tem ganhado cada vez mais destaque, tanto na literatura científica como em novos padrões de comunicação.

É conhecido como decodificação o processo que a partir de uma palavra alterada por ruído determina a palavra original mais provável. Isso consiste em um problema de otimização em que cada bit da palavra código é uma variável $w_i \in \{0, 1\}$, cuja complexidade é $O(2^{n-k})$ para palavras de n bits contendo uma mensagem de k bits, os restantes $m = n - k$ bits são chamados bits de paridade para os códigos LDPC. Uma relaxação desse problema, considerando $w_i \in \mathbb{R}$, permite se aproximar da solução com complexidade $O(n)$ por iteração. A classe de métodos iterativos de decodificação mais conhecidos é chamada de *belief-propagation* e atualizam a estimativa das probabilidades dos bits da palavra código trocando mensagens entre nós correspondentes aos bits da palavra código (*variable-nodes*) e as equações de paridade (*check-nodes*) [5] por meio das arestas do grafo de tanner associado ao código [9].

Alexandre Felipe e André L.N. Souza CPqD, Divisão de Tecnologias Ópticas , Campinas-SP, Brasil, E-mails: afelipe@cpqd.com.br, aluizs@cpqd.com.br. Este trabalho foi financiado pelo FUNTTEL/FINEP e pela FAPESP (2015/25513-6).

A codificação é o processo pelo qual uma mensagem é associada a uma palavra código. Para códigos LDPC a codificação de uma mensagem como uma palavra de um código LDPC tem complexidade $O(m^2)$. Ao longo dos últimos anos a codificação de um código LDPC arbitrário de palavra código grande apresentou-se como um problema menos tratável que a decodificação.

Existem técnicas que permitem codificar eficientemente para alguma estrutura específica de código, e eventualmente um decodificador aceitável existe [8], [4]. Para alcancar o máximo desempenho um código deve ser construído aleatoriamente, evitando ciclos pequenos no grafo de Tanner. Richardson e Urbanke apresentaram um conjunto de heurísticas mostrando que se a grande maioria das linhas da matriz de verificação de paridade possam ser postas de forma triangular apenas com trocas de linhas e colunas, a codificação pode ser realizada com complexidade linear.

A codificação proposta por Richardson e Urbanke requer a resolução de um sistema de equações lineares triangular em campo finito. O problema pode ser resolvido com um número de operações $O(n)$ sequenciais, no entanto, em uma máquina paralela existe uma interdependência entre as operações que tipicamente exige um tempo $O(\sqrt{m})$ para a codificação. O objetivo desse trabalho é indicar maneiras de manipular um sistema de equações parcialmente triangularizado pelo algoritmo de Richardson-Urbanke de modo a reduzir a profundidade do grafo de dependência das operações feitas durante a codificação, permitindo assim implementações paralelas com menor latência.

O restante desse trabalho está organizado da seguinte maneira. A Seção II apresenta uma descrição dos principais conceitos envolvidos no processo de triangularização parcial de matrizes de paridade proposto por Richardson et al. em [7] e a caracterização do problema de latência da codificação usando tais matrizes. Em seguida, a Seção III mostra os resultados de diferentes estratégias de otimização e, por fim, a Seção IV ressalta os pontos mais importantes e conclui o trabalho.

II. Método de Codificação

Assumindo um código com matriz de paridade H de dimensões $m \times n$ e com palavra código w composta por m bits de paridade (p_i) e $n - m$ bits de informação (x_j), de tal forma que $\mathbf{H} \cdot \mathbf{w}^T = \mathbf{H}_m\mathbf{x} + \mathbf{H}_p\mathbf{p} = \mathbf{0}$. O procedimento mais direto de se determinar os bits de paridade é diagonalizar o

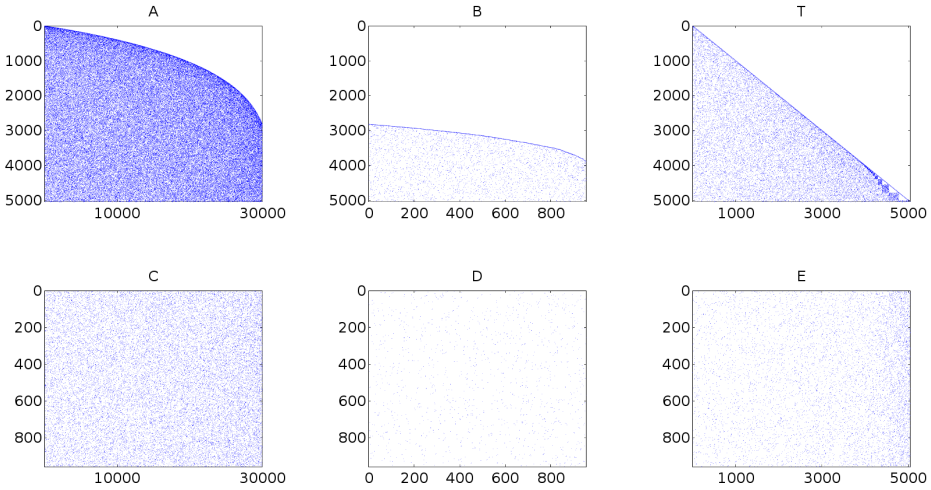

Fig. 1. Particionamento da matriz de verificação de paridade pelo algoritmo CHT de Richardson-Urbanke.

bloco da matriz $\mathbf{H}$ que multiplica os bits de paridade por meio de manipulações de linha $\mathbf{p} = \mathbf{H}_p^{-1} \cdot \mathbf{H}_m \cdot \mathbf{x}$. Essa codificação exigiria $O(m^3)$ operações de preprocessamento e $O(m^2)$ para codificação de uma palavra, já que a matriz $\mathbf{H}_p^{-1}$ não é esparsa.

A. Redução do Número de Operações na Codificação

Técnicas que permitem a codificação eficiente foram propostas por [8], [4] usando grafos cascateados ao invés de grafos bipartidos e outra em [6] forçando a matriz de paridade ser triangular inferior no momento de construção do código, mas ambas acabam reduzindo a capacidade de correção de erros do código gerado. Recentementes códigos LDPC espacialmente acoplados, em que uma mensagem de um código infinito é processada em blocos, permitiram obter um bom desempenho e uma codificação muito simples, no entanto a complexidade de decodificação aumenta consideravelmente, pois para decodificar um bloco é necessário usar uma janela tipicamente com 15 blocos [2].

Richardson e Urbanke apresentam em [7] um algoritmo que permite a codificação com complexidade $O(n)$ por palavra, assumindo matrizes de paridade da forma:

$$H = \begin{bmatrix} A & B & T \\ C & D & E \end{bmatrix}, \tag{1}$$

onde as dimensões de A são $(m-g)\times k$, de B são $(m-g)\times g$, de T são $(m-g)\times(m-g)$, de C são $g\times k$, de D são $g\times g$ e de E são $g\times(n-k-g)$. Todas as submatrizes são esparsas, T é triangular inferior e g é um número pequeno se comparado a m. Para que a codificação tenha complexidade $O(n)$, é necessário que g seja limitado a $O(\sqrt{n})$. Particionando os bits de paridade p como $\begin{bmatrix} p_1^T & p_2^T \end{bmatrix}^T$, p_1 com g bits e p_2 com $(m-g)$ bits, os bits de paridade podem ser calculados como

$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} B & T \\ D & E \end{bmatrix}^{-1} \begin{bmatrix} A \\ C \end{bmatrix} \mathbf{s} = \begin{bmatrix} -\phi^{-1}\theta \\ T^{-1}(B\phi^{-1}\theta + A) \end{bmatrix} \mathbf{s}, \tag{2}$$

com $\phi^{-1} = (-ET^{-1}B + D)^{-1}$ uma matriz densa precomputada, e $\theta = (-ET^{-1}A + C)$.

A obtenção de p_2 pode ser simplificada uma vez computado p_1 da seguinte forma

$$p_1 = -\phi^{-1}(-ET^{-1}A\mathbf{s} + C\mathbf{s}) \tag{3}$$

$$p_2 = -T^{-1}(-Bp_1 + A\mathbf{s}) \tag{4}$$

Richardson e Urbanke também apresentam em [7] alguns métodos gulosos (*Greedy Algorithms*) para triangularizar parcialmente uma matriz esparsa usando apenas permutações de linhas e colunas, não alterando assim o desempenho do código. A principal diferença entre os algoritmos propostos está em como é feita a escolha das linhas e colunas que serão permutadas. Para este trabalho foi escolhido o método serial do algoritmo CHT (Seção III.D de [7]). As submatrizes obtidas com o algoritmo para um código (36,6)-LDPC com matriz de paridade gerada aleatoriamente de tamanho 6000×36000, são ilustradas na Fig. 1.

B. Análise da latência de codificação

Deve-se notar que o formato triangular inferior e a esparsidade da submatriz T torna possível a resolução de um sistema de equações com T como a matriz de coeficientes com complexidade $O(m-g)$. No entanto, esse processo tem estágios não paralelizáveis, *i.e* um bit de paridade só pode ser calculado após o cálculo de todos demais bits que façam parte em pelo menos uma das equações de paridade em que ele aparece. O objetivo desse trabalho é indicar maneiras de manipular um sistema de equações triangular para diminuir o número de estágios não paralelizáveis, reduzindo assim, a latência do sistema de codificação.

Um sistema de equações triangular é uma sequência de relações intrinsecamente ordenadas. Se cada variável for determinada pela equação na qual ela aparece na diagonal, é possível determinar todas, uma a uma sequencialmente, e o número de operações necessárias é o número de elementos não zero na matriz que representa o sistema excluindo a diagonal. Nas implementações em hardware as operações são realizadas imediatamente após todas suas entradas estarem disponíveis, sendo que cada operação ocupa uma área do circuito. A

TABELA I
DETALHAMENTO DE COMPLEXIDADE DE CODIFICAÇÃO EM HARDWARE

Operação	Área	Latência
$[A] \cdot [s]$	$O((m-g)d_c)$	$O(log(d_c))$
$[T^{-1}] \cdot [A\mathbf{s}]$	$O((m-g)(d_v-1))$	$O(\sqrt{(m-g)}\,log(d_c))$
$[E] \cdot [T^{-1}A\mathbf{s}]$	$O(gd_c)$	$O(log(d_c))$
$[C] \cdot [\mathbf{s}]$	$O(gd_c)$	$O(log(d_c))$
$[ET^{-1}A\mathbf{s}] + [C\mathbf{s}]$	O(g)	O(1)
$[\phi^{-1}] \cdot [ET^{-1}A\mathbf{s} + C\mathbf{s}]$	$O(g^2)$	$O(log(g))$
$[B] \cdot [p_1]$	$O((m-g)d_c)$	$O(log(d_c))$
$[T^{-1}] \cdot [Bp_1 + A\mathbf{s}]$	$O((m-g)(d_v-1))$	$O(\sqrt{(m-g)}\,log(d_c))$

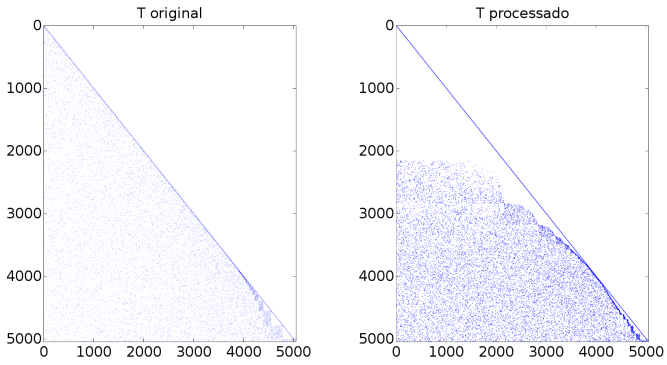

Fig. 2. Resultado das permutações durante o processamento da matriz H com o algoritmo 1.

implementação direta em hardware das relações entre as variáveis do sistema e as variáveis de entrada produz um grafo direcionado acíclico (*directed acyclic graph* - DAG), que pode ser ordenado. O modelo simplificado de hardware consiste em considerar que o resultado de uma operação estará disponível depois de um certo tempo contado a partir do instante em que todas entrada estiverem disponíveis. Considera-se que todas as entradas são disponibilizadas simultaneamente no instante zero, e as demais variáveis terão um atraso determinado pela profundidade de lógica, associada ao comprimento do caminho mais longo até um bit de entrada do sistema.

Duas coisas são importantes no projeto de um circuito em hardware; a latência da lógica e a área em silício ou número de elementos lógicos em FPGA. Se um circuito tem uma profundidade lógica muito grande é necessário diminuir a frequência de relógio, ou permitir que o circuito opere por maior número de ciclos de relógio. A tabela I detalha a complexidade das operações necessárias para a codificação.

O que se observa é que a operação que mais afeta a latência é a multiplicação por T^{-1} que precisa ser feita duas vezes sequencialmente, uma vez para o calculo de p_1 e outra para o cálculo de p_2 que depende de p_1. A operação que mais afeta a área é a multiplicação por ϕ^{-1} que cresce com o quadrado de g. A área do circuito pode ser considerada diretamente proporcional à complexidade computacional em uma máquina sequencial, cuja redução de complexidade já foi estudada em [7].

C. Reduzindo a latência da multiplicação por T^{-1}

Pode-se ordenar as colunas e linhas de T de modo que ela permaneça diagonal e que suas colunas estejam dispostas da menor para a maior distância da variável até a entrada. A matriz ordenada resultante possui termos diferentes de zero na diagonal e imediatamente abaixo possui uma região preenchida com termos zero.

Teorema 1: Se b_i é o número de variáveis que podem ser calculadas com i estágios, e d é o número de estágios necessários para calcular todas as variáveis, P é uma matriz de permutação de modo que o *i-ésimo* bloco de cada linha de PTP' multiplique as variáveis que podem ser calculadas com no mínimo i estágios, e que D_i na diagonal são matrizes diagonais inversíveis de dimensão $b_i \times b_i$, e que $L_{i,(i-1)}$ tenha pelo menos um termo diferente de zero em cada linha.

$$\tilde{T} = PTP' = \begin{bmatrix} D_1 & 0 & 0 & \dots & 0 \\ L_{2,1} & D_2 & 0 & \dots & 0 \\ L_{3,1}, & L_{3,2} & D_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ Ld,1 & L_{d,2} & L_{d,3} & \dots & D_d \end{bmatrix} \quad (5)$$

Demonstração: D_i é diagonal e inversível pois 1. como a matriz $\tilde{T}$ é triangular inferior, todos os blocos da diagonal devem ser triangulares inferiores; 2. Como todas as colunas do bloco i necessitam exatamente i estágios para serem calculados a partir da entrada, essas variáveis devem ser independentes entre si, sendo assim não deve haver mais de um termo em cada linha de um bloco na diagonal; 3. a partir das proposições 1. e 2. a matriz é diagonal, e como $\tilde{T}$ é inversível todas as matrizes na diagonal devem ser inversíveis. Finalmente como as variáveis resolvidas no estágio i dependem de pelo menos uma variável calculada no estágio $i-1$ essa dependência deve aparecer como termos diferentes de zero na matriz $L_{i,(i-1)}$. ■

Lembrando que as colunas da matriz T são parte de uma coluna da matriz de paridade, é possível trocar uma coluna da matriz T por qualquer coluna de A ou B. Se for posível trocar as colunas do bloco i de modo que $L_{(i+1),i}$ tenha alguma linha formada apenas de zeros, a variável resolvida na linha correspondente pode ser calculada independente de qualquer variável do bloco i, reduzindo o número de estágios requeridos para determinar tal variável. Para explorar essa possibilidade é proposto um algoritmo para troca linhas e colunas na matriz H de modo que a estrutura triangular seja mantida e que não afete o valor de g, e que o segundo termo diferente de 0 nas colunas de T estejam o mais distante possível da diagonal.

O Algoritmo 1 troca as colunas da matriz T por outras colunas de H, desde que seja possível aumentar o número de zeros consecutivos imediatamente a baixo da diagonal para uma dada coluna, esse procedimento preserva a distância g, nunca aumenta o número de estágios necessários para determinar uma variável em uma dada coluna e eventualmente pode reduzir o número de estágios requeridos para determinar alguma variável.

```
1  função primeiro(H,j):
2      i ← 1 ;
3      enquanto i ≤ n:
4          se H_{i,j} ≠ 0:
5              retorna i ;
6          i ← i + 1;
7      retorna ∅;
8  função seg(H, j):
9      i ← primeiro(H, j) + 1 ;
10     enquanto i ≤ n:
11         se H_{i,j} ≠ 0:
12             retorna i ;
13         i ← i + 1
14     retorna ∅;
15 para i de 1 até m − g:
16     para j de 1 até k + g:
17         se primeiro(H, j) = i:
18             se seg(H, j) > seg(H, k + g + i):
19                 Permuta as colunas (j) e (k + g + i)
                    da matriz H
```

Algoritmo 1: Algoritmo para otimização da matriz T para codificação paralela

III. Resultados

Nessa seção são apresentados o resultado de redução de latência usando o algritmo de otimização 1. Todos os cálculos foram feitos usando como base um sistema de equações oriundo da manipulação de uma matriz de paridade construída aleatoriamente de tamanho 6000×36000. As submatrizes dessa matriz são mostradas na Fig. 1 após passarem pelo método serial do algoritmo CHT para triangularização parcial. Para o nosso caso, a maior profundidade de lógica de um bit na matriz **H** original é 75 (caminho crítico), como pode-se ver na Fig. 3.

Um tratamento muito simples que pode ser feito com a matriz de paridade **H** é usar o Algoritmo 1. Este procedimento consiste em escolher para compor a submatriz **T** apenas as colunas que tenham o primeiro 1 na linha correta e cujo segundo 1 esteja o mais para baixo possível. Essa simples alteração reduz a ocorrência de variáveis que dependem de outras variáveis em colunas próximas.

O resultado das permutações pode ser visto na Fig. 2. Note uma região branca mais evidente abaixo da diagonal de T, isso significa que existe um grande conjunto de variáveis que podem ser computadas independentemente uma das outras. Consequentemente, o número de nós de profundidade muito pequena é bem maior, e que pode-se esperar em razão dessa característica um caminho crítico menor. Observando a Fig. 3 o número de nós com profundidade menor do que 10 aumentou de 7709 para 8805 e a profundidade de lógica do caminho crítico caiu de 75 para 48, uma redução de 36%.

IV. Conclusão

A redução de latência de codificação se torna um problema crítico em sistemas que operem em taxas muito elevadas (e.g sistemas ópticos trabalhando a taxas acima de 100 Gb/s) e com palavras código longas para otimizar o desempenho total do sistema.

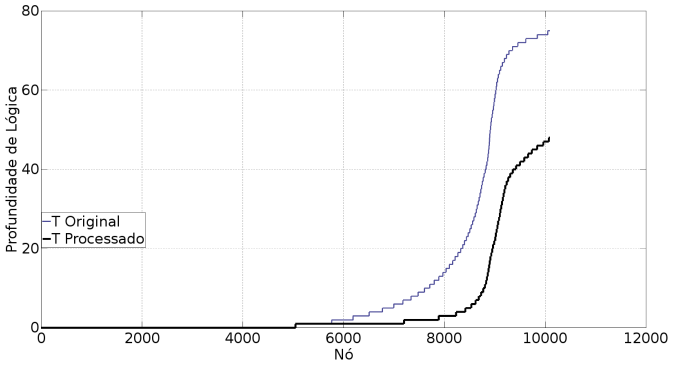

Fig. 3. Profundidade de lógica dos nós ordenados da menor para a maior da matriz T antes (curva azul) e depois do processamento (curva preta).

Com o objetivo de reduzir a latência de codificação de códigos LDPC sem prejudicar o desempenho do código original, esse trabalho propôs um método de permutação de linhas e colunas de forma a reduzir a profundidade de lógicano circuito de resolução sistemas triangulares. Escolher as colunas da matriz H que participarão da submatriz triangular inferior, maximizando a distância entre o primeiro e o segundo 1 para cada coluna individualmente é uma tarefa simples, que pode tanto reduzir a complexidade do sistema para uma máquina sequencial como permitir calcular os bits de paridade em um modelo de computação paralela com um menor número de passos.

Aplicando o novo método em uma matriz de paridade de tamanho 6000×36000 gerada aleatoriamente e parcialmente triangularizada aplicando o algoritmo proposto em [7], foi obtida uma redução de 36% na profundidade de lógica.

Referências

[1] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. 1. In *Communications, 1993. ICC '93 Geneva. Technical Program, Conference Record, IEEE International Conference on*, volume 2, pages 1064–1070 vol.2, May 1993.

[2] Alexandre Felipe, André LN Souza, Juliano RF Oliveira, and Jacklyn D Reis. Scaling design parameters on the overall performance of spatially-coupled ldpc codes. In *Proceedings of the XXXV Brazilian Communications and Signal Processing Symposium*, SBrT '16, 2016.

[3] R. Gallager. Low-density parity-check codes. *IRE Transactions on Information Theory*, 8(1):21–28, January 1962.

[4] Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, Daniel A. Spielman, and Volker Stemann. Practical loss-resilient codes. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 150–159, New York, NY, USA, 1997. ACM.

[5] D. J. C. MacKay and R. M. Neal. Near shannon limit performance of low density parity check codes. *Electronics Letters*, 32(18):1645–, Aug 1996.

[6] D. J. C. MacKay, S. T. Wilson, and M. C. Davey. Comparison of constructions of irregular gallager codes. *IEEE Transactions on Communications*, 47(10):1449–1454, Oct 1999.

[7] T. J. Richardson and R. L. Urbanke. Efficient encoding of low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):638–656, Feb 2001.

[8] M. Sipser and D. A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, Nov 1996.

[9] R. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, Sep 1981.