VLSI Architectures for NB-LDPC Decoders

Xinmiao Zhang

SanDisk Corporation

Binary vs Non-binary (NB) Low-Density Parity-Check (LDPC) Codes

- Binary LDPC codes
  - Require long codeword length
  - Good performance for random errors

- Non-binary LDPC codes
  - Better performance with moderate codeword length
  - Lower error-floor
  - Better at correcting clustered errors

Outline

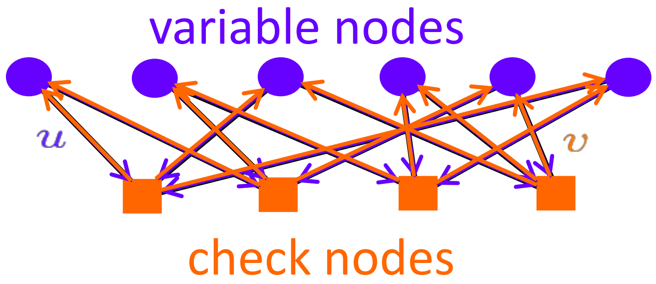
- NB-LDPC codes & design challenges
- NB-LDPC decoding algorithms
- Check node processing architectures
- Overall decoder architectures
- Comparisons & conclusions

LDPC Codes

- LDPC codes are linear block codes, specified by the parity check matrix H
- A received sequence x is a codeword iff $Hx^T = 0$

Binary LDPC code

$$
\begin{array}{c}
\begin{array}{cccccc} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 \end{array} \\
H = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}
\end{array}
$$



check equation

$$x_2 + x_3 + x_4 = 0$$

NB-LDPC Codes

- Nonzero entries of H are elements of $GF(q)$ $(q>2)$

$$H=\begin{bmatrix} \alpha & 0 & \alpha^4 & 0 & 0 & \alpha^5 \\ \alpha^2 & \alpha^3 & 0 & 0 & \alpha & 0 \\ 0 & \alpha^5 & 0 & \alpha & 0 & \alpha^2 \\ 0 & 0 & \alpha^3 & \alpha^4 & \alpha & 0 \end{bmatrix} \dashrightarrow \alpha^3 x_2 + \alpha^4 x_3 + \alpha x_4 = 0$$

Decoder implementation challenges:

Vectors of q messages need to be computed and stored

$\Downarrow$

- large memory requirement
- much more complicated check node processing

NB-LDPC Decoding Algorithms

- Belief propagation (BP)
  - probability-domain: need convolutions
  - frequency-domain: still need many multipliers
  - log-domain: need many look-up tables
  - mixed-domain: need many look-up tables

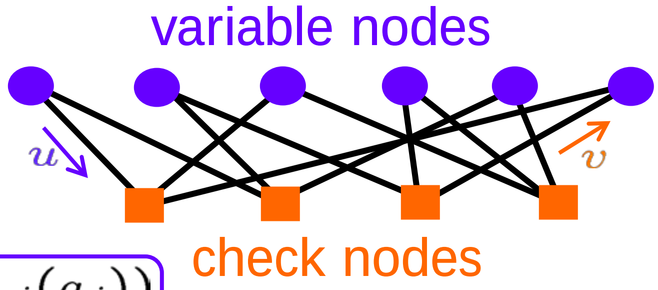
- Extended Min-sum (EMS) algorithm
- Min-max algorithm

Log-domain approximations of BP

Min-max NB-LDPC Decoding Algorithm

messages are represented as log likelihood ratios (LLRs)

$llr(\alpha) = \log(P(z=\hat{\alpha})/P(z=\alpha))$ $\hat{\alpha}$: most likely finite field element



Initialization: $u_{m,n}(\alpha) = \gamma_n(\alpha)$

Iterations:

- Check node processing

$$\boxed{v_{m,n}(\alpha) = \min_{(a_j)\in\mathcal{L}(m|a_n=\alpha)} \left(\max_{j\in S_v(m)\backslash n} u_{m,j}(a_j) \right)}$$

- Variable node processing

$$u_{m,n}(\alpha) = \gamma_n(\alpha) + \textstyle\sum_{i\in S_c(n)\backslash m} v_{i,n}(\alpha)$$

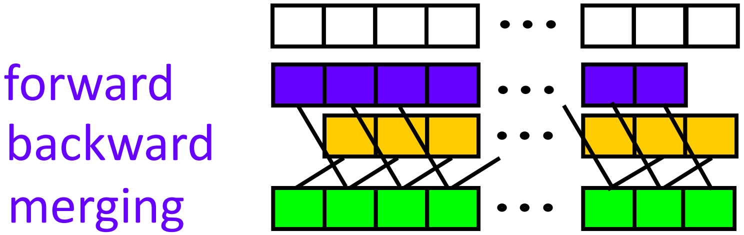
- A posteriori information computation

$$\tilde{\gamma}_n(\alpha) = \gamma_n(\alpha) + \textstyle\sum_{i\in S_c(n)} v_{i,n}(\alpha)$$

Min-max Check Node Unit (CNUs)

- Forward-backward
- Path-construction based
- Simplified Min-max
- Basis-construction based
- Modified trellis-based using syndromes

Forward-backward Check Node Processing



Elementary step:

$$f_i(\alpha) = \min_{\alpha' + \alpha'' = \alpha} \left(\max\left(f_{i-1}(\alpha'), u_{m,n_i}(\alpha'') \right) \right)$$

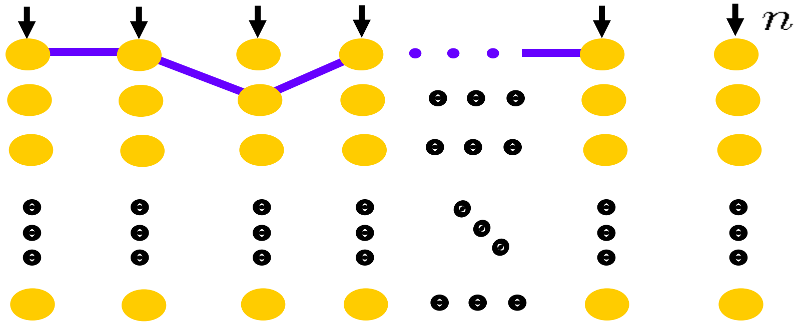
Disadvantages:

- large number of intermediate results need to be stored
- Large number of recursive computations

Trellis Representation of Messages

$$v_{m,n}(\alpha) = \min_{(a_j)\in\mathcal{L}(m|a_n=\alpha)} \left(\max_{j\in S_v(m)\setminus n} u_{m,j}(a_j) \right)$$

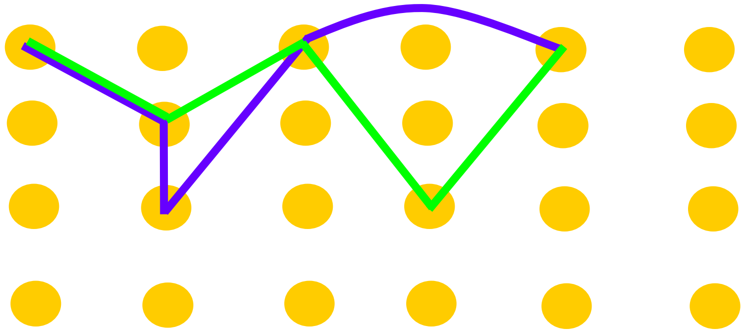
v2c message vectors



- (a_j) corresponds to a path that passes exactly one node in each stage, except the stage for variable node n
- Computing $v_{m,n}$ is equivalent to finding the paths with the smallest LLRs and different finite field elements

Relaxations on Path Configuration

- Relaxation: multiple nodes in a path can come from the same stage



- A node can be considered as an approximation of the node with the same field element from another stage
- The over or under-estimated LLR does not have much noticeable effect on the Min-max decoding performance

Basis-construction Check Node Processing

$\{\omega_1, \omega_2, \ldots, \omega_p\}$ is a basis of $GF(2^p)$

any $\alpha \in GF(2^p)$ can be written as a linear combination of ω_i

$\Downarrow$

$v_{m,n}(\alpha)$ can be computed *parallely* from nodes in minimum basis B_j

p nodes $\notin$ stage j
with minimum nonzero LLRs
& independent field elements

- Using the relaxation, the construction of B_j can be greatly simplied
- Each B_j can be derived by updating a global basis with $p+n_c$ entries

Check Node Processing Using Syndromes

$$v_{m,n}(\alpha) = \min_{(a_j)\in\mathcal{L}(m|a_n=\alpha)} \left(\max_{j\in S_v(m)\setminus n} u_{m,j}(a_j) \right)$$

alternate approach

- Compute syndromes $w(\alpha) = \min_{(a_j)\in\mathcal{T}(m|\alpha)} \left(\max_{j\in S_v(m)} u_{m,j}(a_j) \right)$
- Take out the contribution of the nodes in stage n from the syndrome to derive c2v messages

$$\hat{v}_{m,n}(\alpha - \eta_n^{(\alpha)}) = \min(\hat{v}_{m,n}(\alpha - \eta_n^{(\alpha)}), w(\alpha) - \hat{u}_{m,n}(\eta_n^{(\alpha)}))$$

element of stage n in (a_j) leading to $w(\alpha)$

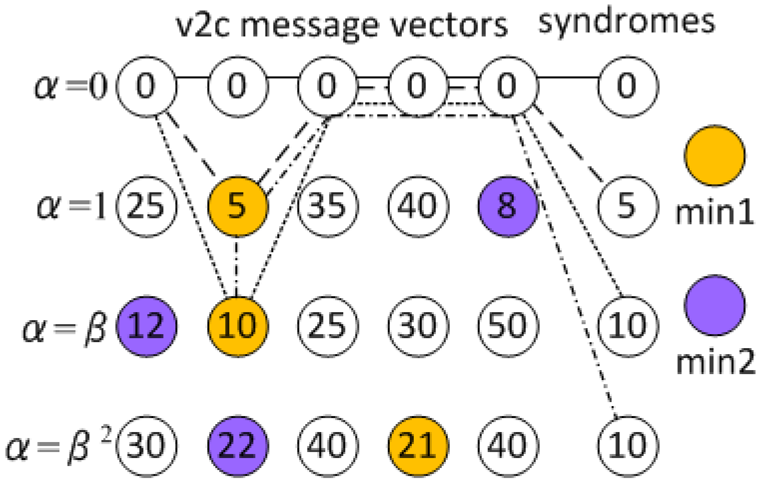
Modified Syndrome Computation: GF(4) Code

transformed trellis

$$\hat{u}_{m,n}(\alpha) = u_{m,n}(\alpha + \hat{\alpha})$$

β: a primitive element of GF(4)

$$1 + \beta = \beta^2$$



- Need only one max comparator for each syndrome

Modified Message Recovery from Syndromes

- $\hat{v}_{m,n}(\alpha)$ for $n \in S_v(m)$ are recovered from $w(\alpha)$

inputs: $w(\alpha)$, $\min1(\alpha)$, $\mathrm{idx}(\alpha)$, $\min2(\alpha)$, # of deviation nodes in $w(\alpha)$ path
for each $\alpha \neq 0$
If there is one deviation node, and it is in stage i:

$$\hat{v}_{m,n}(\alpha) = \begin{cases} min1(\alpha) \text{ if } n \neq i \\ min2(\alpha) \text{ if } n = i \end{cases}$$

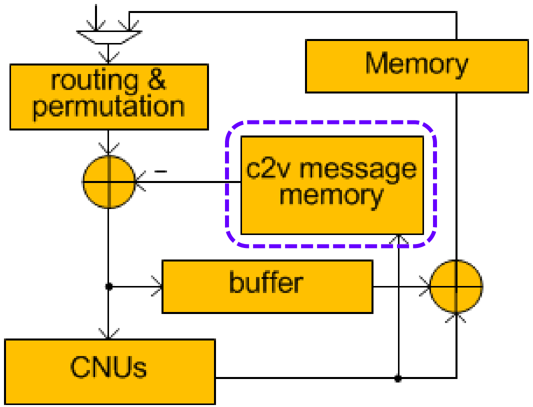
If there are two deviation nodes, and they are in stages i and j:

$$\hat{v}_{m,n}(\alpha) = \begin{cases} w(\alpha) \text{ if } n \neq i, j \\ min1(\alpha) \text{ if } n = i \text{ or } j \text{ and } n \neq idx(\alpha) \\ min2(\alpha) \text{ if } n = i \text{ or } j \text{ and } n = idx(\alpha) \end{cases}$$

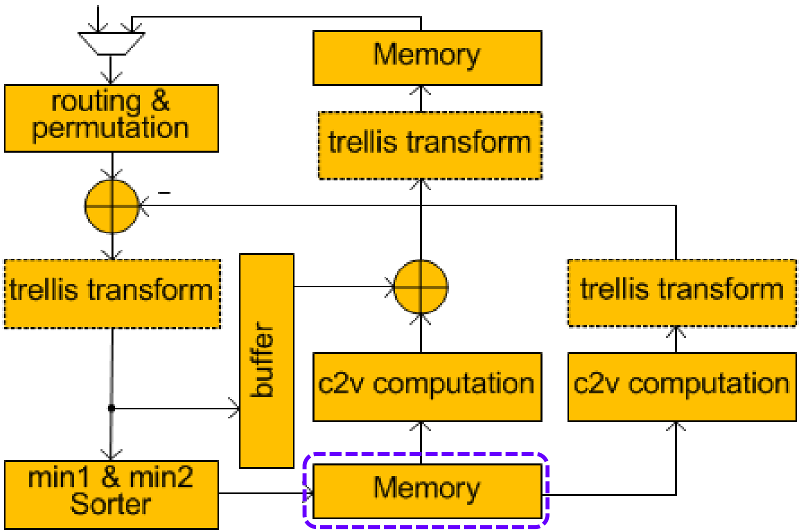
- Implementable by simple index testers and multiplexors

Layered Quasi-cyclic Decoder Architectures

Decoder with forward-backward CNU



Decoder with CNUs using min1, min2



- smaller memory
- more efficient CNUs

Comparisons & Conclusions: GF(q) Codes

A. Forward-backward

- Not efficient

B. Path construction

- Intra-vector serial computation
- Can keep $< q$ messages, memory advantage for larger q

C. Simplified Min-max

- Intra-vector parallel computation
- Complexity close to D and E for small q; less efficient for larger q

D. Basis construction

- Intra-vector parallel computation
- May have smaller area than E for larger q

E. Modified Trellis-based using syndromes

- Intra-vector parallel computation; enable efficient inter-vector parallel processing
- Most efficient for GF(4) codes

Questions?

xinmiao.zhang@sandisk.com