Nonbinary LDPC Decoding and its Implementation

Jie Huang

November 5, 2008
Outline

- Definition
- Decoding Algorithms
- C Implementation
- FPGA Implementation
- Conclusions
Example of nonbinary LDPC code

$$Hx = 0$$

$$H = \begin{bmatrix}
5 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 3 & 0 & 0 & 3 & 0 & 0 & 5 & 0 & 0 & 3 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 5 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 5 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 4 & 0 \\
0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 4 & 0 & 4 & 0 & 7 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 4 & 6 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 6 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

- A LDPC code over GF(8)
- The nonzero entries can take values from \{1,2,\ldots,7\}
Tanner graph representation

\[ Hx = 0 \]

(a) \( H \); (b) its Tanner graph.

- Variable nodes correspond to columns of \( H \).
- Check nodes correspond to rows of \( H \).
Decoding for nonbinary LDPC codes

- Direct implementation has complexity on the order of $O(q^2)$ for GF($q$) [Davey-Mackay'98].

- FFT-based SPA has complexity on the order of $O(q \log q)$ in [Song-etal'03] and [Barnault-etal'03].

- Log domain decoding without multiplication/division operations in [Wymeersch-etal’04].

- Approximate version, like min-sum and extended min-sum.
Messages representation

- $GF(q) = \{\alpha_0, \alpha_1, \ldots, \alpha_{q-1}\}$ where $\alpha_0 = 0$.

- Probability messages
  - $[p(v = \alpha_0), p(v = \alpha_1), \ldots, p(v = \alpha_{q-1})]^T$

- Log domain messages
  - $L^+(v) = [L(v = \alpha_0), L(v = \alpha_1), \ldots, L(v = \alpha_{q-1})]^T$
Tentative decoding

- \( q_j^a = \alpha_j f_j^a \prod_{m \in \mathcal{M}(j)} r_{mj}^a \)
- \( \alpha_j \) is chosen such that \( \sum_{a=0}^{q-1} q_j^a = 1 \)
- \( \mathcal{M}(j) \equiv \{ i : H_{ij} \neq 0 \}, \mathcal{N}(i) \equiv \{ j : H_{ij} \neq 0 \} \)
Variable node update

- $q_{ij}^a = \alpha_{ij} f_j^a \prod_{m \in M(j) \setminus i} r_{mj}^a$

- $\alpha_{ij}$ is chosen such that $\sum_{a=0}^{q-1} q_{ij}^a = 1$
Check node update

\[ \mathcal{N}(i) \equiv \{ j : H_{ij} \neq 0 \} \]

\[ r_{ij}^{a} = \sum_{x : x_j = a} \delta \left[ H_{ij} a + \sum_{j' \in \mathcal{N}(i) \setminus j} H_{ij'} x_{j'} \right] \prod_{j' \in \mathcal{N}(i) \setminus j} q_{ij'}^{x_{j'}} \]
**FFT-based updating rule**

- Suppose that $\mathcal{N}(i) = \{j_1, j_2, j_3\}$

- $H_{ij_1} x_{j_1} + H_{ij_2} x_{j_2} + H_{ij_3} x_{j_3} = 0$

- Given the distribution for $x_{j_2}$ and $x_{j_3}$, how to compute the distribution for $x_{j_1}$?

- $r_{ij_1}^a = \sum_b q_{ij_2}^b q_{ij_3}^{H_{ij_3}^{-1}(H_{ij_2} b + H_{ij_1} a)}$

- Denote $H_{ijk} x_{jk} = y_k, k = 1, 2, 3,$ then $y_1 + y_2 + y_3 = 0$

- $Pr(-y_1 = a) = \sum_b Pr(y_2 = b) Pr(y_3 = a - b)$
FFT-based decoding

- The updating rules at check node and variable node are the same, except that there is no channel information at the check node.
- For GF(2^p), the FFT is a p-dimension FFT, each dimension is of size 2.
- From my Ph.D thesis.
Message flow for FFT based decoding

\[
\gamma_{ij}^a = q_{ij}^{H_{ij}^{-1}a},
\]

\[\text{Do } p\text{-dimension FFT over } \gamma_{ij}^a \text{ to obtain } F\gamma_{ij}^a,\]

\[F\xi_{ij}^a = \prod_{n \in \mathcal{N}(i) \setminus j} F\gamma_{in}^a,\]

\[\text{Do } p\text{-dimension IFFT over } F\xi_{ij}^a \text{ to obtain } \xi_{ij}^a,\]

\[r_{ij}^a = \xi_{ij}^{H_{ij}a},\]

\[q_{ij}^a = \alpha_{ij} f_j^a \prod_{m \in \mathcal{M}(j) \setminus i} r_{mj}^a,\]
Messages representation

- \( GF(q) = \{ \alpha_0, \alpha_1, \ldots, \alpha_{q-1} \} \) where \( \alpha_0 = 0 \).

- Probability messages
  - \( [p(v = \alpha_0), p(v = \alpha_1), \ldots, p(v = \alpha_{q-1})]^T \)

- Log domain messages
  - \( L^+(v) = [L(v = \alpha_0), L(v = \alpha_1), \ldots, L(v = \alpha_{q-1})]^T \)
BoxPlus operation

- $L^\dagger(A_1 v_1 + A_2 v_2) := \oplus(L_1^\dagger, L_2^\dagger, A_1, A_2)$, where $A_1 \in GF(q)$ and $A_2 \in GF(q)$, $v_1$ and $v_2$ are random variables over $GF(q)$.

- $L^\dagger(A_1 v_1 + A_2 v_2)_i = \ln \left( \sum_{x \in GF(q)} e^{L_1^\dagger(x) + L_2^\dagger(A_2^{-1}(\alpha_i - A_1 x))} \right) - \ln \left( \sum_{x \in GF(q)} e^{L_1^\dagger(x) + L_2^\dagger(-A_2^{-1}A_1 x)} \right)$

- $max^*(x, y) := \ln(e^x + e^y) = max(x, y) + \ln(1 + e^{-|x-y|})$

- min-sum
Log domain variable node update

\[ L^+_{post}(x_j) = L^+_{ch}(x_j) + \sum_{k \in M(j)} L^+_{\leftarrow}(k \rightarrow j) \]

\[ L^+_{\leftarrow}(i \leftarrow j) = L^+_{ch}(x_j) + \sum_{k \in M(j)/i} L^+_{\leftarrow}(k \rightarrow j) \]

\[ \text{component-wise summation} \]
Log domain check node update

1. Suppose $j_{i,0} < j_{i,1} < \cdots < j_{i,|\mathcal{N}(i)|-1}$, define
   \[
   \sigma_{i,j_i,k} = \sum_{l \leq k} H_{i,j_i,l} x_{j_i,l} \quad \text{and} \quad \rho_{i,j_i,k} = \sum_{l \geq k} H_{i,j_i,l} x_{j_i,l},
   \]
   $k = 0, \ldots, |\mathcal{N}(i)| - 1$

2. \[
   L^\dagger(\sigma_{i,j_i,k}) = L^\dagger(\sigma_{i,j_i,k-1} + H_{i,j_i,k} x_{j_i,k})
   \]

3. \[
   L^\dagger(\rho_{i,j_i,k}) = L^\dagger(\rho_{i,j_i,k+1} + H_{i,j_i,k} x_{j_i,k})
   \]

4. \[
   L^\dagger(i \rightarrow j_i,k) = L^\dagger(-H_{i,j_i,k}^{-1}(\sigma_{i,j_i,k-1} + \rho_{i,j_i,k+1}))
   \]
Log domain check node update illustration

- Forward and backward recursion
C Implementation

- Double-linked list to store the H matrix
- FFT based implementation and log domain implementation
- `typedef struct { //Representation of a sparse matrix
  int n_rows; //number of rows
  int n_cols; //number of columns
  int field_index; //finite field index
  double **f; //information from the channel
  mod2entry *rows; //Pointer to array of row headers
  mod2entry *cols; //Pointer to array of column headers
  mod2block *blocks; //Blocks that have been allocated
  mod2entry *next_free; //Next free entry
} mod2sparse;`
C Implementation

- typedef struct mod2block {
  //Block of entries
  struct mod2block *next; //Next block that has been allocated
  mod2entry entry[10]; //Entries in this block
} mod2block;

- typedef struct mod2entry {
  //a nonzero entry
  int row, col; //Row and column indexes of this entry
  struct mod2entry *left, *right, *up, *down; //pointers
  double *q; //Messages exchanged during the decoding
  int value; //The value of the nonzero entry
} mod2entry;
FPGA Implementation

- Area/Resource
- Throughput
- Energy consumption
- Storage/Memory
- Connections/Layers
- Quantization effect/Overflow
Architecture

- Parallel, all rows followed by all columns and so on.

- Sequential, row-by-row or column-by-column

- Partially parallel, group-by-group, column-wise or row-wise

- row-by-row sequential decoding can save memory storage.
Parallel updating BP decoding

TIME
Sequential updating BP decoding
Partially parallel updating BP decoding
Comparison of different updating schedule

- The average number of iterations of sequential BP can be about half that of parallel BP.

- The complexity per iteration for both parallel BP and sequential BP is similar, resulting in a lower total complexity for sequential BP.

- Low throughput is the main drawback of sequential BP.

- Partially parallel BP offers better throughput/complexity tradeoffs in the implementation of efficient decoders.
FFT-based or log domain?

- FFT-based algorithm has much less complexity

- Log domain algorithm requires fewer quantization levels and suffers less from a quantization induced error-floor.
  - “computational complexity and quantization effects of decoding algorithms for non-binary LDPC codes” by H. Wymeersch, et al.

- Could we combine both?
Binary parallel implementation example

“A 220mW 1Gb/s 1024-Bit Rate-1/2 LDPC Code Decoder” by C. Howland and A. Blanksby.

The main challenge was the placement and routing of the macros at the top level with more than 26000 wires of average length 3mm representing the messages.
Conclusions

- Decoding complexity is the main concern for nonbinary LDPC codes
- FPGA based implementation is a challenging job
- High throughput/low power consumption is desirable