Generic model of LDPC code decoders

Frédéric GUILLOUD
Emmanuel BOUTILLON
guilloud@enst-bretagne.fr
emmanuel.boutillon@univ-ubs.fr
Motivation

- Find « generic parameters » to characterize a LDPC architecture.
- (Try to) classify existing architectures
- Propose new architectures
Hypothesis 2

- Regular (j,k) LDPC code
- Belief Propagation decoding algorithm
- Check node in the frequency domain

\[ E_{m,n} = I_n + \sum_{m' \in M(n) \setminus m} T_{m',n} \]

\[ T_{m,n} = \Phi^{-1} \left( \sum_{n' \in N(m) \setminus n} \Phi(E_{m,n'}) \right) \]

\[ \Phi(x) = -\ln \left( \tanh \left( \frac{x}{2} \right) \right) \]
OUTLINE

- Motivation
- Architecture parameters
  - Parallelism
  - Scheduling
  - Type of generic node (VNU, GNU)
  - Position of the interconnection network
- Example of architecture
- Conclusion
Parallelism of a \((j,k) = (3,6)\) LDPC

\(P_c\) : computational power: number of edges computed/cycle

\(K\) number of information bit

\(R\) rate of the code

\(E\) number of edge of the code:

\(F_{\text{clock}}\) Clock frequency:

\(F_{\text{bit}}\) Bit throughput:

\(N_{\text{it}}\) average number of iteration

\[
P_c = E N_{\text{it}} \left[ \text{edge/codeword} \right] \times \left( \frac{F_{\text{bit}}}{K} \right) \frac{1}{F_{\text{clock}}} \quad \text{[codeword/clock cycle]}
\]

\[
= \frac{3K}{0.5} \times 20 \times \frac{10}{K} \times \frac{1}{100} = 12 \quad \text{[edge/clock cycle]}
\]
Serial/Parallel solution

Several solutions

$\alpha_v = 1$ cycles

$\alpha_v = 3$ cycles

$\alpha_c = 1$ cycles

$\alpha_c = 2$ cycles

$\alpha_v = 1$ cycles

$\alpha_v = 3$ cycles

$\alpha_v = 1$ cycles

$\alpha_v = 3$ cycles

$\alpha_v = 1$ cycles

$\alpha_v = 3$ cycles

$\alpha_v = 1$ cycles

$\alpha_v = 3$ cycles

$\alpha_v = 1$ cycles

$\alpha_v = 3$ cycles

$\alpha_c = 1$ cycles

$\alpha_c = 2$ cycles

$\alpha_c = 3$ cycles
Generic model

• $n_v, n_c$: number of input/output ports of the node units

• $\alpha_c, \alpha_v$: number of clock cycles required to process a CN and a VN

\[
P_c = n_c \times P = n_v \times V
\]
OUTLINE

- Motivation
- Architecture parameters
  - Parallelism
  - Scheduling
  - Type of generic node (VNU, GNU)
  - Position of the interconnection network
- Example of architecture
- Conclusion
Flooding scheduling

Compact scheduling for the check
Flooding scheduling

Compact scheduling for the check
Distributed scheduling for the variable, slow update
Flooding scheduling

Compact scheduling for the check
Distributed scheduling for the variable, slow update
Flooding scheduling

Compact scheduling for the check
Distributed scheduling for the variable, slow update
Flooding scheduling

Compact scheduling for the check
Distributed scheduling for the variable, slow update
Flooding scheduling

Compact scheduling for the check
Distributed scheduling for the variable, slow update
Horizontal scheduling (turbo-scheduling)

LDPC Matrix

Compact scheduling for the check
Horizontal scheduling (turbo-scheduling)

LDPC Matrix

Compact scheduling for the check
Distributed scheduling for the variable, fast update
Horizontal scheduling (turbo-scheduling)

LDPC matrix

Compact scheduling for the check
Distributed scheduling for the variable, slow update

http://lester.univ-ubs.fr/
Horizontal scheduling (turbo-scheduling)

Matrice LDPC

Number of iteration divided by 1/2

Compact scheduling for the check
Distributed scheduling for the variable, slow update

E. Boutillon
http://lester.univ-ubs.fr/
Vertical scheduling (Fossorier)

LDPC matrix

Compact scheduling for the check
Distributed scheduling for the variable, slow update
Vertical scheduling (Fossorier)

<table>
<thead>
<tr>
<th>011110</th>
<th>101101</th>
<th>110101</th>
</tr>
</thead>
</table>

LDPC Matrix

Distributed scheduling for the check, fast update
Compact scheduling for the variable
Vertical scheduling (Fossorier)

LDPC Matrix

Distributed scheduling for the check, fast update
Compact scheduling for the variable
**Vertical scheduling (Fossorier)**

**LDPC matrix**

```
011110
101101
110101
```

Number of iteration divided by 1/2

Distributed scheduling for the check, fast update
Compact scheduling for the variable

http://lester.univ-ubs.fr/
OUTLINE

- Motivation
- Architecture parameters
  - Parallelism
  - Scheduling
  - Type of generic node (VNU, GNU)
  - Position of the interconnection network
- Example of architecture
- Conclusion

E. Boutillon

http://lester.univ-ubs.fr/
Computation of a variable node

\[ t = i_n + \sum e_m \]

\[ a_i = i_n + \sum e_i = t - a_i \quad \text{for} \quad m, m \neq i \]
Variable node: data flow graph
Computation of a parity check node

\[ e_i = \bigoplus_{j \neq i} a_i = a_1 \oplus a_2 \ldots \oplus a_{i-1} \oplus a_{i+1} \ldots \oplus a_{dc} \]

Two methods  
=> exact: probability or frequency domain  
=> approximated: min-sum and its variants
Parity check node in the frequency domain (sign)
Parity check node in the frequency domain (module)

\[ |e_k| = \Phi^{-1}\left( \sum_{i \neq j} \Phi(|a_k|) \right) \]

\[ \Phi(x) = -\ln(\tanh(x/2)) \]

Generic Node Unit

http://lester.univ-ubs.fr/
Generic Node Units (exclude the Φ function)

- $d$ inputs $e_m$ and $d$ outputs $s_n$ ($d=j$ or $k$)
- Input / output law:

$$s_n = \sum_{m \neq n} e_m$$
Generic Node

a) Compact mode («no memories »)

Compact mode and parallel implementation
($\alpha = 1 \rightarrow n=d$)

Compact mode and serial implementation
($\alpha = d \rightarrow n = 1$)
b) Distributed Mode, slow update (Serial)

\[ t^{(i)} = e_1^{(i)} + e_2^{(i)} + ... + e_d^{(i)} \]

\[ t^{(i+1)} = e_1^{(i+1)} + ... + e_n^{(i+1)} \]
Generic Node:

c) Distributed Mode, fast update

\[ t^{(i+1)}_{n-1} = e_1^{(i+1)} + e_{n-1}^{(i+1)} + e_n^{(i)} + \ldots + e_d^{(i)} \]

\[ t_n^{(i+1)} = e_1^{(i+1)} + e_{n-1}^{(i+1)} + e_n^{(i+1)} + \ldots + e_d^{(i)} \]

Permutation network

\[ t^{(i+1)} \text{ is changing all along the iteration} \]
OUTLINE

- Motivation
- Architecture parameters
  - Parallelism
  - Scheduling
  - Type of generic node (VNU, GNU)
  - Position of the interconnection network
- Example of architecture
- Conclusion
Position of the interconnexion network

\[
\begin{align*}
\Phi - \Sigma & \rightarrow \Phi^{-1}(x) \\
\Sigma & \rightarrow \Phi^{-1}(x) \\
\Sigma & \rightarrow \Phi^{-1}(x) \\
\Phi - \Sigma & \rightarrow \Phi^{-1}(x)
\end{align*}
\]

or min sum…

<table>
<thead>
<tr>
<th>Network position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic Node</td>
<td>VNU</td>
<td>Σ</td>
<td>Φ \circ \Sigma</td>
<td>Σ \circ \Phi^{-1}</td>
</tr>
<tr>
<td></td>
<td>CNU</td>
<td>Φ^{-1} \circ \Sigma \circ \Phi</td>
<td>Φ^{-1} \circ \Sigma</td>
<td>Σ \circ \Phi</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\times (sign product)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Result: Generic description of the decoder architecture

- Node Unit Architecture
- Parallelism: \( P, V, \alpha_{CNU}, \alpha_{VNU} \)
- Processeur control (compact / distributed)
- Shuffle network position (1, 2, 3, 4)

Many combinations of the parameters are possible:
## Control mode combinations

<table>
<thead>
<tr>
<th></th>
<th>VNU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Distributed</td>
</tr>
<tr>
<td>Compact</td>
<td>Slow Update</td>
</tr>
<tr>
<td>Compact</td>
<td>Fast Update</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Compact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Parallel Flooding</td>
</tr>
<tr>
<td></td>
<td>Flooding along CNU</td>
</tr>
<tr>
<td></td>
<td>Horizontal Shuffle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Slow Update</td>
</tr>
<tr>
<td></td>
<td>Fast Update</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Compact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Flooding along VNU</td>
</tr>
<tr>
<td></td>
<td>Edge by edge control</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Vertical Shuffle</td>
</tr>
</tbody>
</table>
OUTLINE

- Motivation
- Architecture parameters
- Example of architecture
- Conclusion
Vertical Shuffle ad-hoc architecture (1)

- Originally proposed as Shuffle-BP (Zhang & Fossorier, 2002)

- Combination of:
  - Compact Mode for VNU
  - Distributed Mode with fast update for CNU
Vertical Shuffle ad-hoc architecture (1)

\[
E_{m,n} = I_n + \sum_{m'\in M(n)\setminus m} T_{m',n}
\]

\[
T_{m,n} = \Phi^{-1}\left( \sum_{n'\in N(m)\setminus n} \Phi(E_{m,n'}) \right)
\]

\[
\Phi(x) = -\ln\left(\tanh\left(\frac{x}{2}\right)\right)
\]

\[
Q_{m,n} = \Phi(E_{m,n'}) \quad R_m = \sum_{n\in N(m)} \Phi(E_{m,n}) \quad T_{m,n} = \Phi^{-1}\left( R_n - Q_{m,n} \right)
\]
Vertical Shuffle ad-hoc architecture (2)

- Compact, Serial VNU

\[
T_{n,m}(i) + \sum_{m' \in M(n) \setminus m} E_{m,n}^{(i-1)}
\]

\[
I_n = \sum_{m' \in M(n) \setminus m} E_{m,n}^{(i-1)}
\]

\[
E_{m_1,n}^{(i-1)}
\]

\[
E_{m_2,n}^{(i-1)}
\]

\[
E_{m_3,n}^{(i-1)}
\]

\[
T_{n,m_1}(i)
\]

\[
T_{n,m_2}(i)
\]

\[
T_{n,m_3}(i)
\]
Vertical Shuffle ad-hoc architecture (3)

Go through the $\phi$ functions and interleaver

$$Q_{n,m} = \Phi(E_{m,n})$$
Vertical Shuffle ad-hoc architecture (4)

- Save $Q_{n,m}$ and update $R_m$ (fast update)

\[
\Phi \left( \sum_{m' \in M(n) \backslash m} E^{(i-1)}_{m,n} \right) + T^{(i)}_{n,m} + I_n = Q^{(i)}_{n,m}
\]

\[
Q^{(i)}_{n,m} - Q^{(i-1)}_{n,m} = R^{(i)}_m
\]
Vertical Shuffle ad-hoc architecture (5)

\[ R_m = \sum_{n \in N(m)} \Phi(E_{m,n}) \]

\[ Q_{n,m} \]

\[ Q_{n,m}^{(i)} - Q_{n,m}^{(i-1)} \]

\[ R_{m,n}^{(i)} \]

\[ R_{m,n}^{(i-1)} \]

\[ E_{m,n}^{(i-1)} \]

\[ T_{n,m}^{(i)} \]

Serial GNU Compact Mode

\[ \Phi \]

interleaver
OUTLINE

- Motivation
- Architecture parameters
- Example of architecture
- Conclusion
Conclusion

Proposition of partial framework for description, analysis and synthesis of LDPC decoder architectures.

Based on the formalism, proposition of an architecture for vertical scheduling

Extend methodology to all LDPC architecture (to be done)
Thank you for your attention!