innovation for industry

LDPC codes and Message Passing decoders

– a brief review –

Valentin Savin

July 2013

Outline

PART 1:

LDPC codes and Message Passing (MP) decoders – brief review

PART 2:

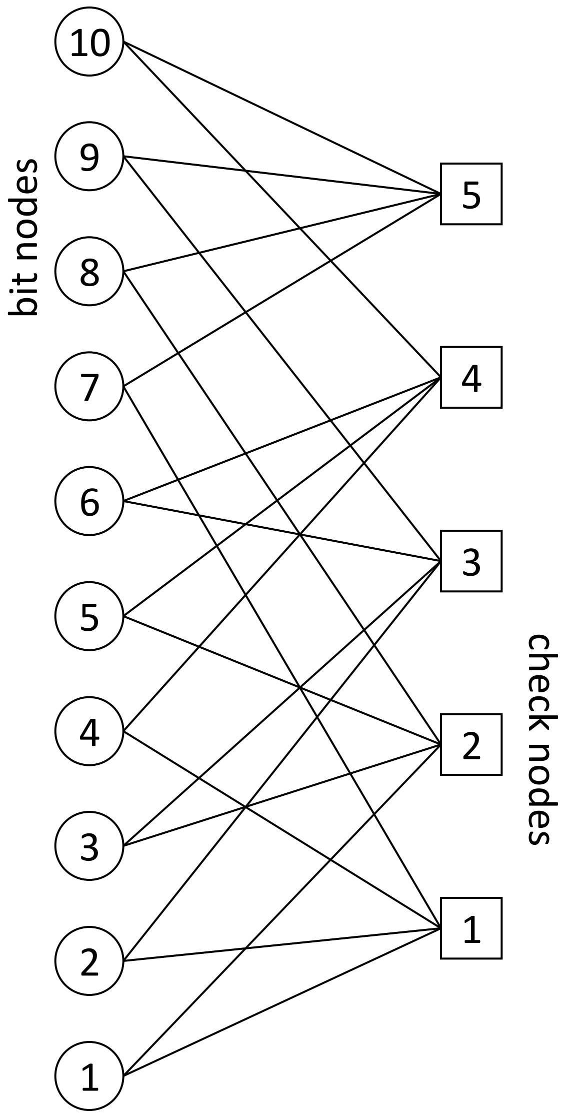
Fixed-point implementation and "noisy" decoders

Graphical representation of LDPC codes

LDPC code

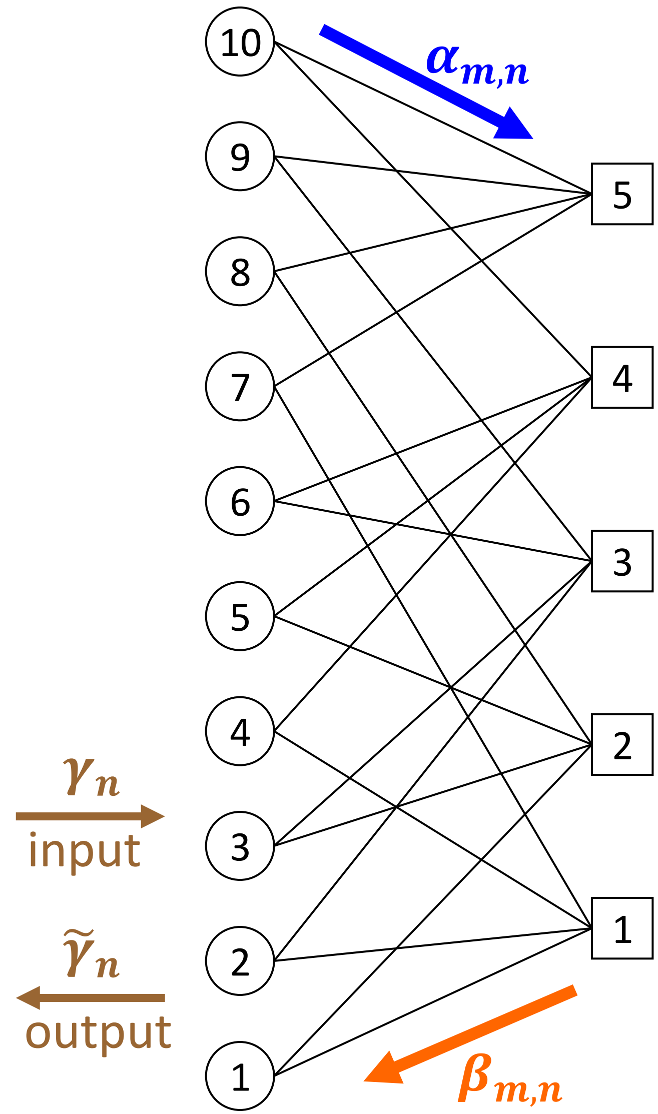
- Linear block code defined by a sparse parity-check matrix H
- H is advantageously represented by a *bipartite (Tanner) graph*
  - circles: *variable-nodes* (or *bit-nodes*) representing coded bits (columns of H)
  - squares: *check-nodes* representing or parity-check equations (rows of H)
  - Graph *edges* correspond to the non-zero entries of H
- Codeword: vector $(x_1, \ldots, x_N)$ such that $H * (x_1, \ldots, x_N)^T = 0$
  - The sum modulo 2 of bits connected to any check-node is zero

$$
\begin{array}{cc}
 & \begin{array}{cccccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} \end{array} \\
H = & \begin{pmatrix}
1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{pmatrix}
\end{array}
$$



Message-Passing (MP) decoders

- Iterative exchange of messages between:
  - variable-nodes ($n = 1, \dots, 10$, on the left), and
  - check-nodes ($m = 1, \dots, 5$, on the right)
- Extrinsic-information exchange
  - $\beta_{m,n} = \mathrm{funct}\left(\alpha_{m,n'};\ n' \in H(m) \backslash n\right)$
  - $\alpha_{m,n} = \mathrm{funct}\left(\gamma_n, \beta_{m',n};\ m' \in H(n) \backslash m\right)$
- A posteriori information
  - $\tilde{\gamma}_n = \mathrm{funct}\left(\gamma_n, \beta_{m,n};\, m \in H(n)\right)$
  - hard-decision is taken based on $\tilde{\gamma}_n$ values



Remark: γ_n = input LLR; $\tilde{\gamma}_n$ = output (AP-)LLR

Message-Passing (MP) decoders

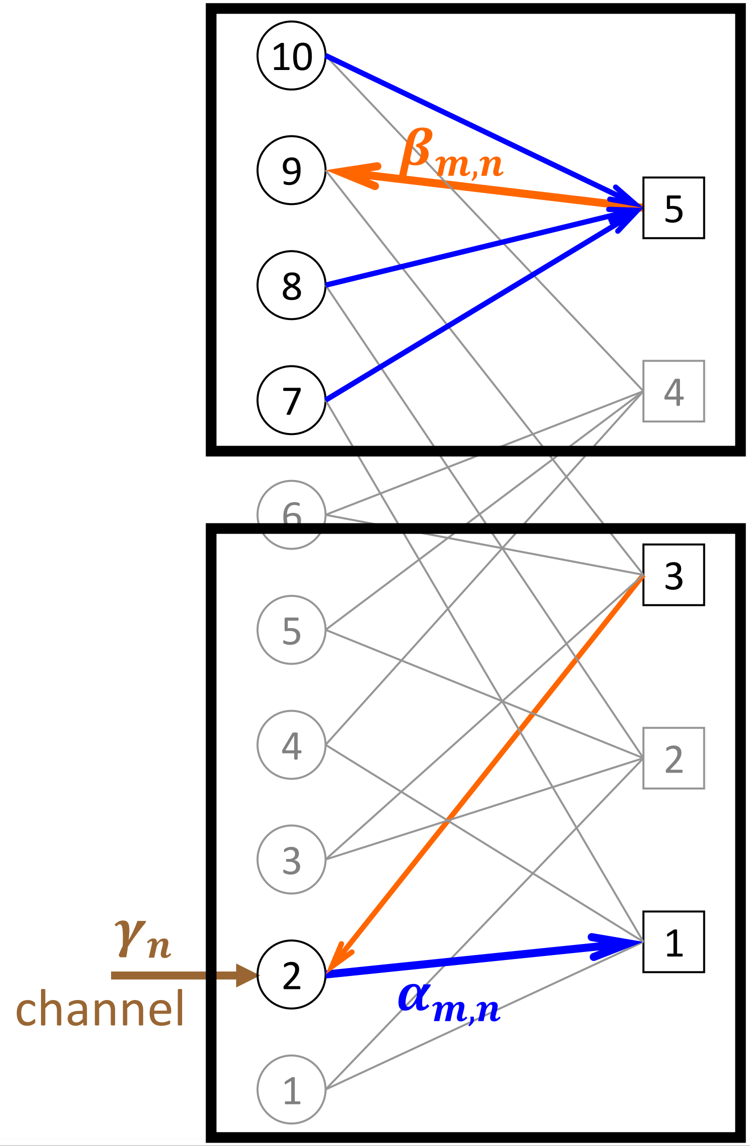
- Extrinsic-information exchange

CNU (Check-Node Unit)

- $\beta_{m,n} = \text{funct}\left(\alpha_{m,n'};\ n' \in H(m)\backslash n\right)$

VNU (Variable-Node Unit)

- $\alpha_{m,n} = \text{funct}\left(\gamma_n, \beta_{m',n};\ m' \in H(n)\backslash m\right)$



Message-Passing (MP) decoders

- Several MP decoders, distinguished by the formulae used to compute check and variable-node messages
  - Gallager A, Gallager-B
  - Sum-Product (or Belief-Propagation)
  - Min-Sum

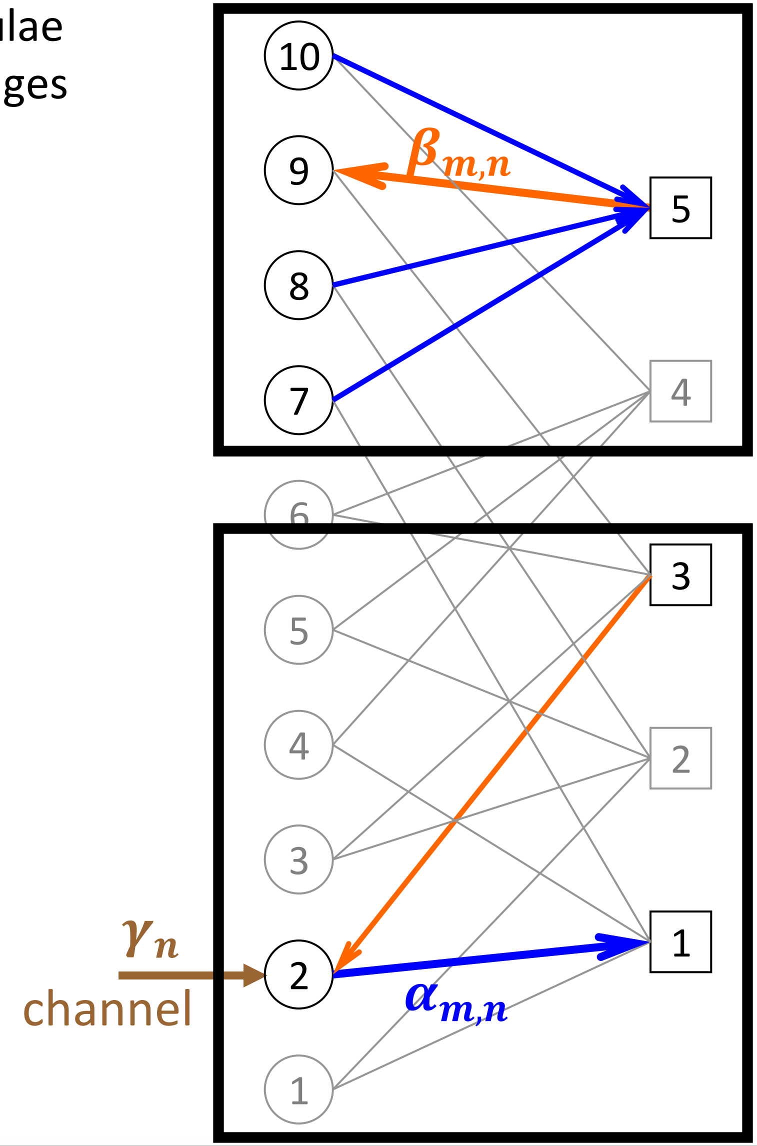
- **Sum-Product decoding**

$$\beta_{m,n} = \left(\prod_{n' \in H(m)\setminus n} \mathrm{sgn}(\alpha_{m,n'})\right) \phi\left(\sum_{n' \in H(m)\setminus n} \phi(|\alpha_{m,n'}|)\right)$$

where $\phi(x) = \log\left(\frac{e^x+1}{e^x-1}\right)$

$$\alpha_{m,n} = \gamma_n + \sum_{m' \in H(n)\setminus m} \beta_{m',n}$$

The outgoing $\alpha_{m,n}$ message is the sum on channel LLR and incoming $\beta_{m,n}$ messages



Message-Passing (MP) decoders

- Several MP decoders, distinguished by the formulae used to compute check and variable-node messages
  - Gallager A, Gallager-B
  - Sum-Product (or Belief-Propagation)
  - Min-Sum

- **Min-Sum decoding**

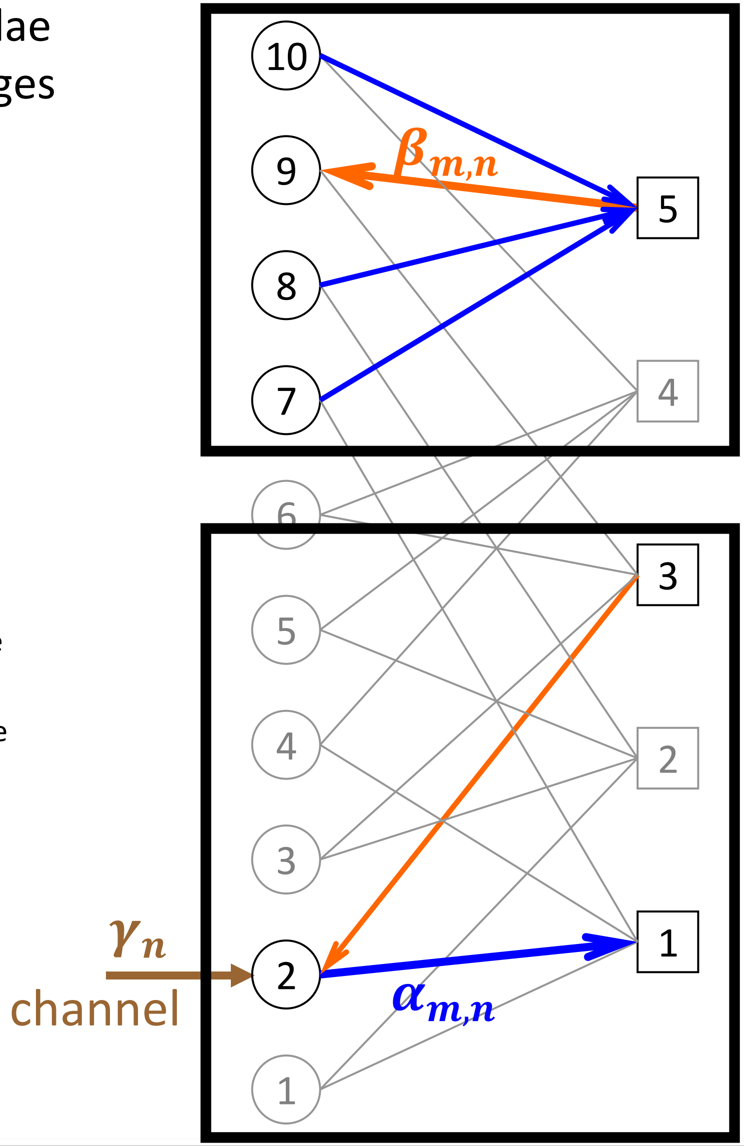
$$\beta_{m,n} = \left(\prod_{n' \in H(m) \backslash n} \operatorname{sgn}(\alpha_{m,n'}) \right) \min_{n' \in H(m) \backslash n} (|\alpha_{m,n'}|)$$

The sign of the outgoing $\beta_{m,n}$ message is the product of the signs of the incoming $\alpha_{m,n'}$ messages

The absolute value of the outgoing $\beta_{m,n}$ message is the minimum of the absolute values of the incoming $\alpha_{m,n'}$ messages

$$\alpha_{m,n} = \gamma_n + \sum_{m' \in H(n) \backslash m} \beta_{m',n}$$

The outgoing $\alpha_{m,n}$ message is the sum on channel LLR and incoming $\beta_{m,n}$ messages



Message-Passing (MP) Scheduling

- Order in which messages are passed between check and variable nodes
- Also determines the order in which CNUs and VNUs are processed

- **Flooding(*) scheduling**

  At each decoding iteration:

  - all CNUs are processed and all check-node messages ($\boldsymbol{\beta_{m,n}}$) are sent from check to variable-nodes
  - then all VNUs are processed, and all variable-node messages ($\boldsymbol{\alpha_{m,n}}$) are sent from variable to check-nodes

  (*) variable and check-nodes are "*flooded*" by incoming messages

- **Layered scheduling**

  - parity check matrix is partitioned in horizontal layers

  At each decoding iteration:

  - all check nodes in one layer are processed and outgoing messages ($\boldsymbol{\beta_{m,n}}$) are sent to their neighbor variable nodes
  - variable-nodes connected to check-nodes in the layer are "updated" (explained later on)
  - move to next layer

Min-Sum decoding with flooding scheduling

Initialization: $\forall n = 1, \dots, N;\ \forall m \in H(n)$

$$\gamma_n = \log(\Pr(x_n = 0 \mid y_n)/\Pr(x_n = 1 \mid y_n))$$
$$\alpha_{m,n} = \gamma_n$$

variable-node messages are initialized acc. to channel LLRs

Iterations: for iter = 1,…, iter_max

- **CNU**: $\forall m = 1, \dots, M;\ \forall n \in H(m)$

$$\beta_{m,n} = \left(\prod_{n' \in H(m) \setminus n} \operatorname{sgn}(\alpha_{m,n'}) \right) \min_{n' \in H(m) \setminus n} (|\alpha_{m,n'}|)$$

all check-nodes are processed
new check-node messages are sent to neighbor var.-nodes

- **VNU**: $\forall n = 1, \dots, N;\ \forall m \in H(n)$

$$\alpha_{m,n} = \gamma_n + \sum_{m' \in H(n) \setminus m} \beta_{m',n}$$

all variable-nodes are processed
new var.-node messages are sent to neighbor check nodes

- **AP-LLR**: $\forall n = 1, \dots, N$

$$\tilde{\gamma}_n = \gamma_n + \sum_{m \in H(n)} \beta_{m,n}$$

a posteriori LLRs of variable-nodes are computed

N.B: these values are used to take a hard decision on each coded-bit (given by the sign of the AP-LLR)

Remark: at each iteration the decoder also computes the syndrome of the estimated word (determined by the signs of the AP-LLR values). Decoder stops if a codeword has been found (syndrome = 0), or if the maximum number of iterations (iter_max) has been reached.

Min-Sum decoding with flooding scheduling

Initialization: $\forall n = 1, \dots, N;\ \forall m \in H(n)$

$$\gamma_n = \log(\Pr(x_n = 0 \mid y_n)/\Pr(x_n = 1 \mid y_n))$$
$$\alpha_{m,n} = \gamma_n$$

variable-node messages are initialized acc. to channel LLRs

Iterations: for iter = 1,…, iter_max

- **CNU**: $\forall m = 1, \dots, M;\ \forall n \in H(m)$

$$\beta_{m,n} = \left(\prod_{n' \in H(m)\setminus n} \operatorname{sgn}(\alpha_{m,n'})\right) \min_{n' \in H(m)\setminus n} (|\alpha_{m,n'}|)$$

all check-nodes are processed
new check-node messages are sent to neighbor var.-nodes

- **AP-LLR**: $\forall n = 1, \dots, N$

$$\tilde{\gamma}_n = \gamma_n + \sum_{m \in H(n)} \beta_{m,n}$$

a posteriori LLRs of variable-node are computed

N.B: these values are used to take a hard decision on each coded-bit (given by the sign of the AP-LLR)

- **VNU**: $\forall n = 1, \dots, N;\ \forall m \in H(n)$

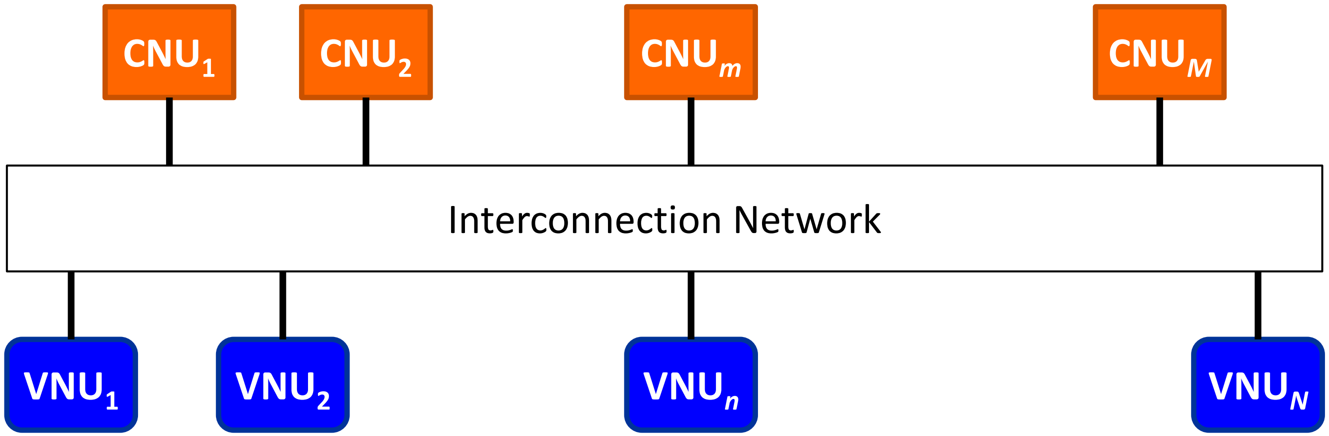
$$\alpha_{m,n} = \tilde{\gamma}_n - \beta_{m,n}$$

all variable-nodes are processed
new var.-node messages are sent to neighbor check nodes

AP-LLR and VNU steps can be advantageously switched
(they can actually be merged in a single processing unit)

Remark: at each iteration the decoder also computes the syndrome of the estimated word (determined by the signs of the AP-LLR values). Decoder stops if a codeword has been found (syndrome = 0), or if the maximum number of iterations (iter_max) has been reached.

MP decoding with flooding scheduling



one CNU for each check-node in the Tanner graph

one VNU for each bit-node in the Tanner graph

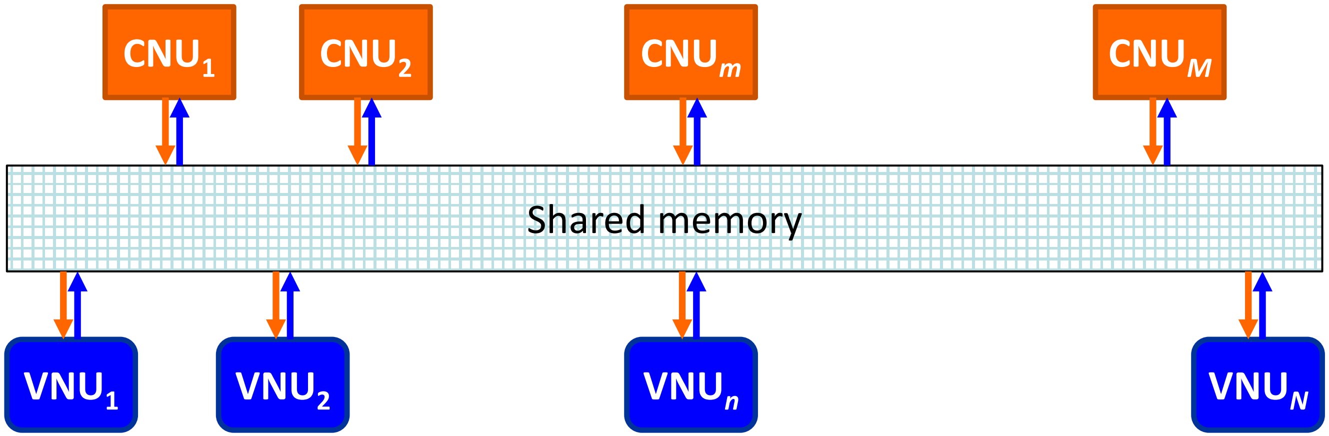
- First half iteration: all CNUs are processed
  - **processed in parallel** if the system allows parallel computation
  - otherwise, **processed sequentially**
- Second half iteration: all VNUs are processed
  - **processed in parallel** if the system allows parallel computation
  - otherwise, **processed sequentially**
- Main issue: complexity of the interconnection network

fully parallel decoder

sequential decoder

but the same flooding scheduling!

MP decoding with flooding scheduling



one CNU for each check-node in the Tanner graph

one VNU for each bit-node in the Tanner graph

- First half iteration: all CNUs are processed
  - **processed in parallel** if the system allows parallel computation
  - otherwise, **processed sequentially**
- Second half iteration: all VNUs are processed
  - **processed in parallel** if the system allows parallel computation
  - otherwise, **processed sequentially**
- Main issue: dealing with memory conflicts

fully parallel decoder

sequential decoder

but the same flooding scheduling!

MP decoding with layered scheduling

- The parity check matrix H is partitioned in L horizontal layers
  - Each layer contains μ = M/L rows
    - Layer l is determined by μ consecutive rows: $\mathcal{M}_l = (l-1)\mu + 1, \dots, l\mu$
  - Matrix H is usually designed such that any variable-node is connected at most once to any layer (e.g. Quasi-Cyclic LDPC codes).

<u>**Decoding scheduling (how it works):**</u>

$$H = \begin{array}{c} \left(\begin{array}{ccccc} 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 1 & 0 \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \hline 0 & 1 & 0 & \cdots & 1 & 1 \end{array}\right) \end{array} \begin{array}{l} \mathcal{M}_1 \text{ (μ rows)} \\ \mathcal{M}_2 \text{ (μ rows)} \\ \mathcal{M}_3 \text{ (μ rows)} \end{array}$$

Process check nodes in the first layer, then update all bit-nodes connected to the layer

Move to the next layer: process check nodes in the layer, then update all bit-nodes connected to this layer

Continue with following layers…

then start a new decoding iteration

Min-Sum decoding with layered scheduling

Initialization: $\forall n = 1, \dots, N;\ \forall m \in H(n)$

$$\tilde{\gamma}_n = \gamma_n = \log(\Pr(x_n = 0 \mid y_n)/\Pr(x_n = 1 \mid y_n))$$
$$\beta_{m,n} = 0$$

AP-LLR values are initialized from channel LLR values
check-node messages are initialized to 0

Iterations: for iter = 1,…, iter_max

- **Loop over horizontal layers**: $\forall\ l = 1, \dots, L$
  - **VNU**: $\forall m \in \mathcal{M}_l;\ \ \forall n \in H(m)$

    $$\alpha_{m,n} = \tilde{\gamma}_n - \beta_{m,n}$$

    for each check-node in the current layer, incoming variable-to-check ($\alpha_{m,n}$) messages are computed by subtracting the $\beta_{m,n}$ message from the AP-LLR value

  - **CNU:** $\forall m \in \mathcal{M}_l;\ \ \forall n \in H(m)$

    $$\beta_{m,n} = \left(\prod_{n' \in H(m) \backslash n} \operatorname{sgn}(\alpha_{m,n'})\right) \min_{n' \in H(m) \backslash n} (|\alpha_{m,n'}|)$$

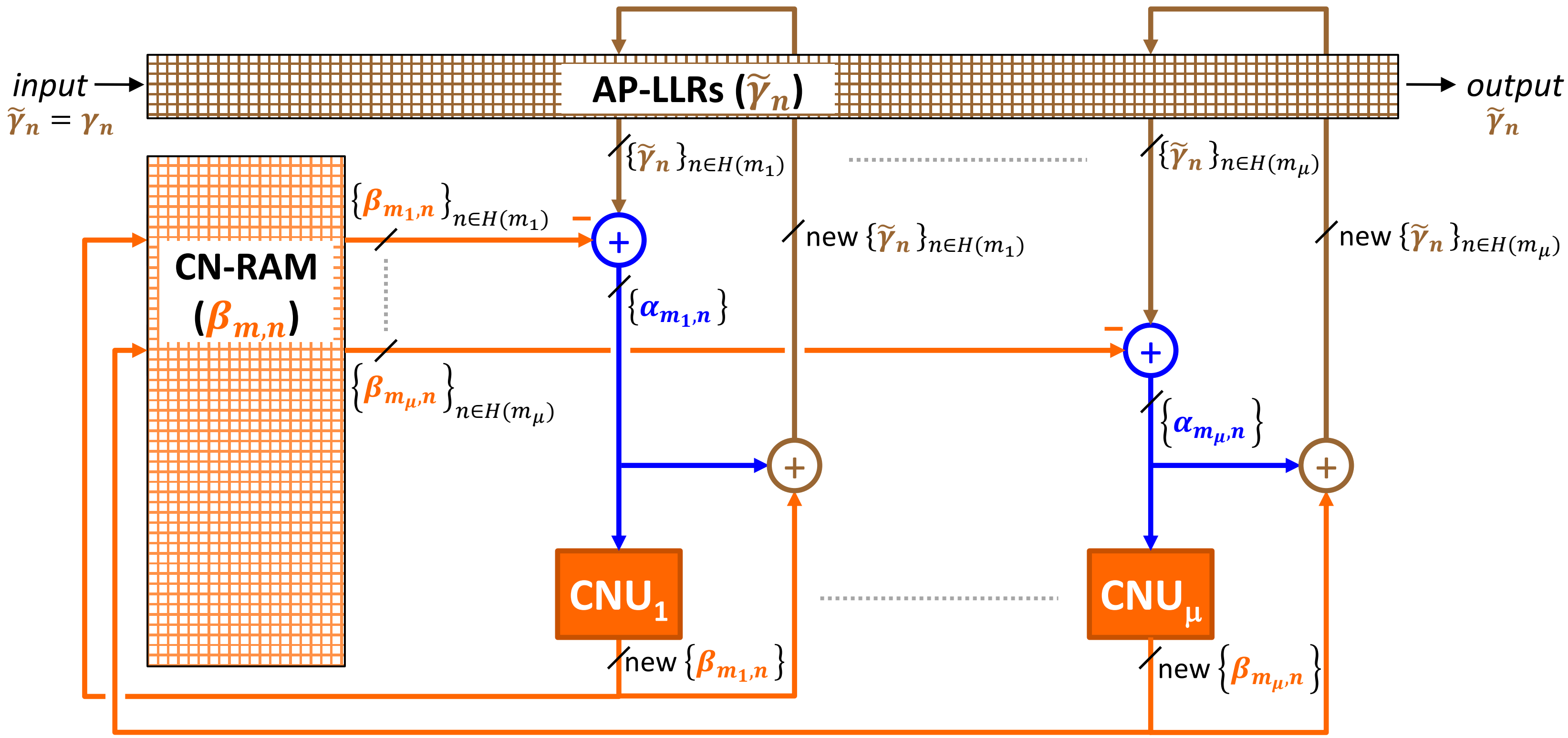
    check-nodes in the current layer are processed
    new check-to-variable messages $\beta_{m,n}$ are computed

  - **AP-LLR**: $\forall m \in \mathcal{M}_l;\ \ \forall n \in H(m)$

    $$\tilde{\gamma}_n = \alpha_{m,n} + \beta_{m,n}$$

    a posteriori LLR values are updated according to the new check-to-variable ($\beta_{m,n}$) messages

MP decoding with layered scheduling



- Number of CNU = number of for check-node per layer
  - **processed in parallel** if the system allows parallel computation
  - otherwise, **processed sequentially**
- No memory conflicts, provided that any variable-node is connected to at most one check-node in each layer

partially parallel decoder

sequential decoder

but the same
layered scheduling!

MP decoding with layered scheduling

Interest of layered scheduling:

- Simple architecture, with a flexible degree of parallelism
- Converges (twice) faster than flooding scheduling
  - better performance, if the maximum number of decoding iterations is small

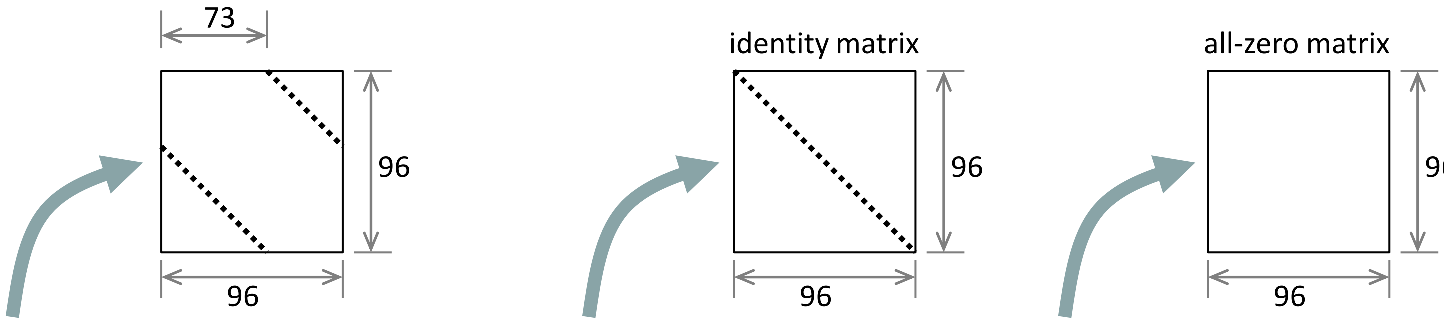
Faster convergence:

- AP-LLR values of variable-nodes connected to a layer are updated after the check-nodes in that layer have been processed
- When the next layer is processed, the incoming check-node messages ($\alpha_{m,n}$) are derived from the AP-LLR values of the corresponding variable-nodes .
- Hence, incoming messages incorporate the contribution of the check-nodes from the previous layers (even those processed in the same iteration)
- This yield a faster propagation (*) of messages through the decoding graph, thus speeding up the decoder convergence

(*) It can be proved that messages propagate twice faster through cycle-free graphs
In practice, the propagation speed up factor is less than twice…

Example-1: WiMAX LDPC code with rate 1/2

- Parity-check matrix *H* of size 1152 x 2304
- **Quasi-Cyclic** code: *H* is obtained by expanding a *base matrix B* of size 12 x 24
  - each entry of *B* is expanded to a square matrix of size 96 x 96
  - a non-negative entry *b* is replaced by a cyclic permutation matrix = identity right-shifted by *b* columns



B =

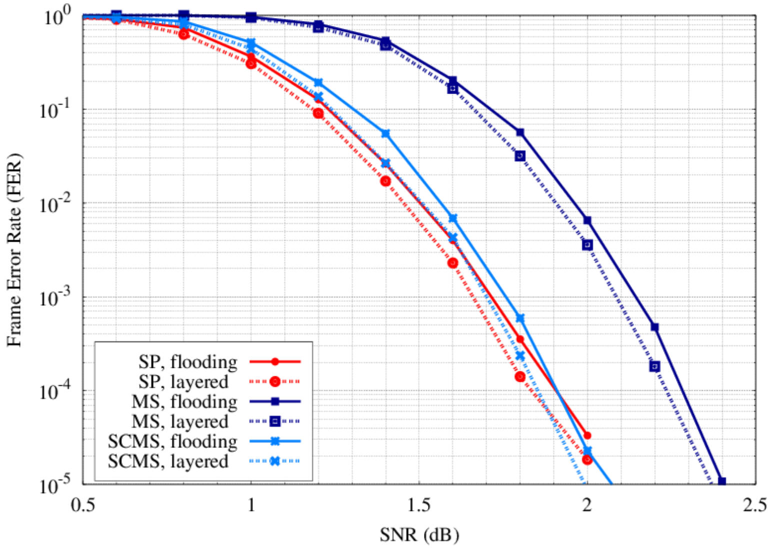
-1	94	73	-1	-1	-1	-1	-1	55	83	-1	-1	7	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	27	-1	-1	-1	22	79	9	-1	-1	-1	12	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	24	22	81	-1	33	-1	-1	-1	0	-1	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1
61	-1	47	-1	-1	-1	-1	-1	65	25	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1	-1
-1	-1	39	-1	-1	-1	84	-1	-1	41	72	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	46	40	-1	82	-1	-1	-1	79	0	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1
-1	-1	95	53	-1	-1	-1	-1	-1	14	18	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1
-1	11	73	-1	-1	-1	2	-1	-1	47	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1	-1
12	-1	-1	-1	83	24	-1	43	-1	-1	-1	51	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1
-1	-1	-1	-1	-1	94	-1	59	-1	-1	70	72	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1
-1	-1	7	65	-1	-1	-1	-1	39	49	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
43	-1	-1	-1	-1	66	-1	41	-1	-1	-1	26	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0

← Layer 1 (96 rows)
← Layer 2 (96 rows)

(when layered scheduling is used)

← Layer 12 (96 rows)

Example-1: WiMAX LDPC code with rate 1/2

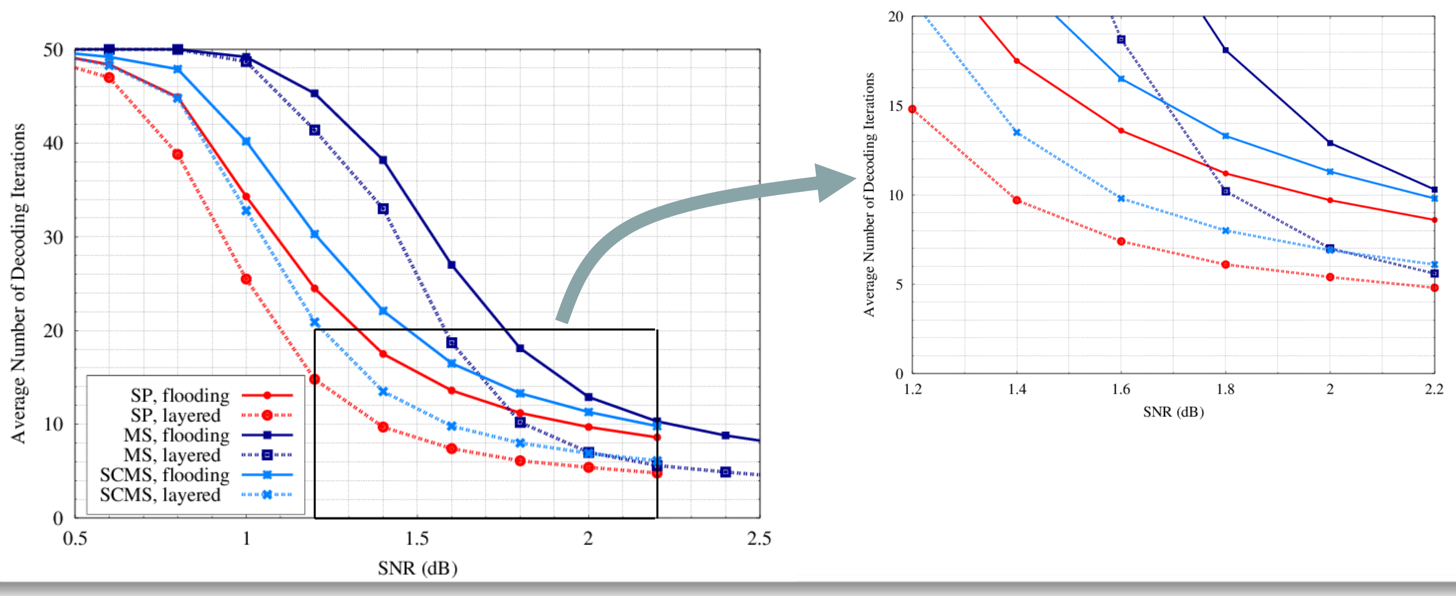


AWGN, QPSK
Maximun number of decoding iterations = 50

Color code:

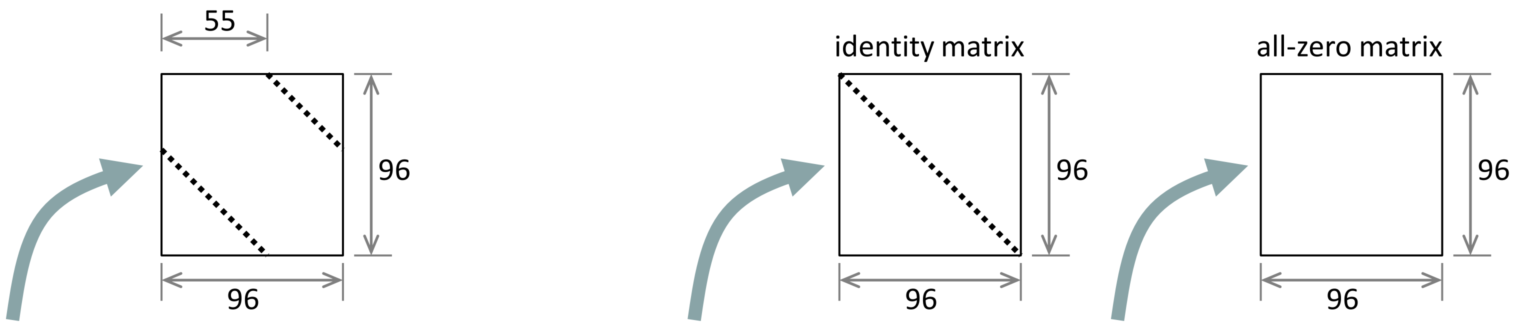
- **Sum-Product** (**SP**)
- **Min-Sum** (**MS**)
- **Self-Corrected Min-Sum** (**SCMS**)

Solid curves : flooding scheduling
Dashed curves: layered scheduling



Example-2: WiMAX LDPC code with rate 5/6

- Parity-check matrix *H* of size 384 x 2304
- **Quasi-Cyclic** code: *H* is obtained by expanding a *base matrix B* of size 4 x 24
  - each entry of *B* is expanded to a square matrix of size 96 x 96
  - a non-negative entry *b* is replaced by a cyclic permutation matrix = identity right-shifted by *b* columns



B =

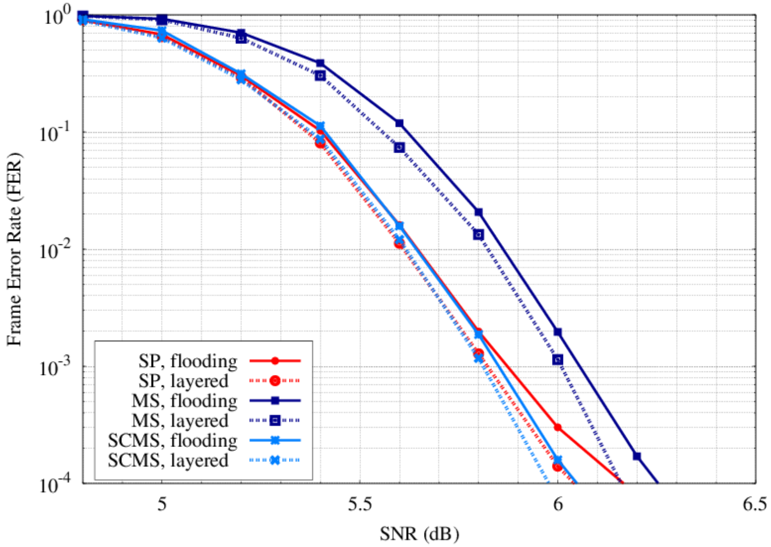
1	25	55	-1	47	4	-1	91	84	8	86	52	82	33	5	0	36	20	4	77	80	0	-1	-1
-1	6	-1	36	40	47	12	79	47	-1	41	21	12	71	14	72	0	44	49	0	0	0	0	-1
51	81	83	4	67	-1	21	-1	31	24	91	61	81	9	86	78	60	88	67	15	-1	-1	0	0
50	-1	50	15	-1	36	13	10	11	20	53	90	29	92	57	30	84	92	11	66	80	-1	-1	0

← Layer 1 (96 rows)

(when layered scheduling is used)

← Layer 4 (96 rows)

Example-2: WiMAX LDPC code with rate 5/6

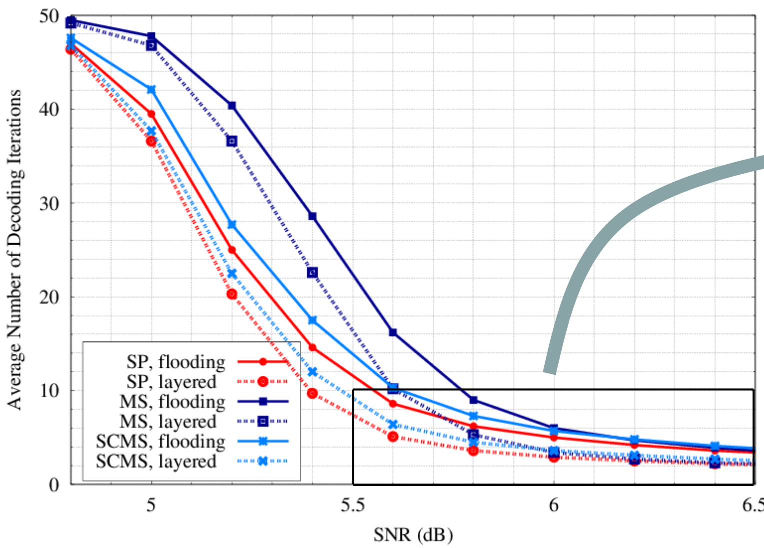


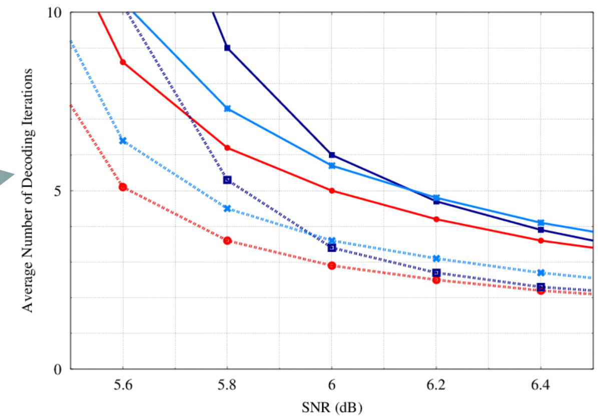
AWGN, QPSK
Maximun number of decoding iterations = 50

Color code:

- **Sum-Product** (**SP**)
- **Min-Sum** (**MS**)
- **Self-Corrected Min-Sum** (**SCMS**)

Solid curves : flooding scheduling
Dashed curves: layered scheduling





Outline

PART 1:

LDPC codes and Message Passing (MP) decoders – brief review

PART 2:

Fixed-point implementation and "noisy" arithmetic units

Impact of the scheduling and of the quantization of the decoder performance
1. Noiseless devices

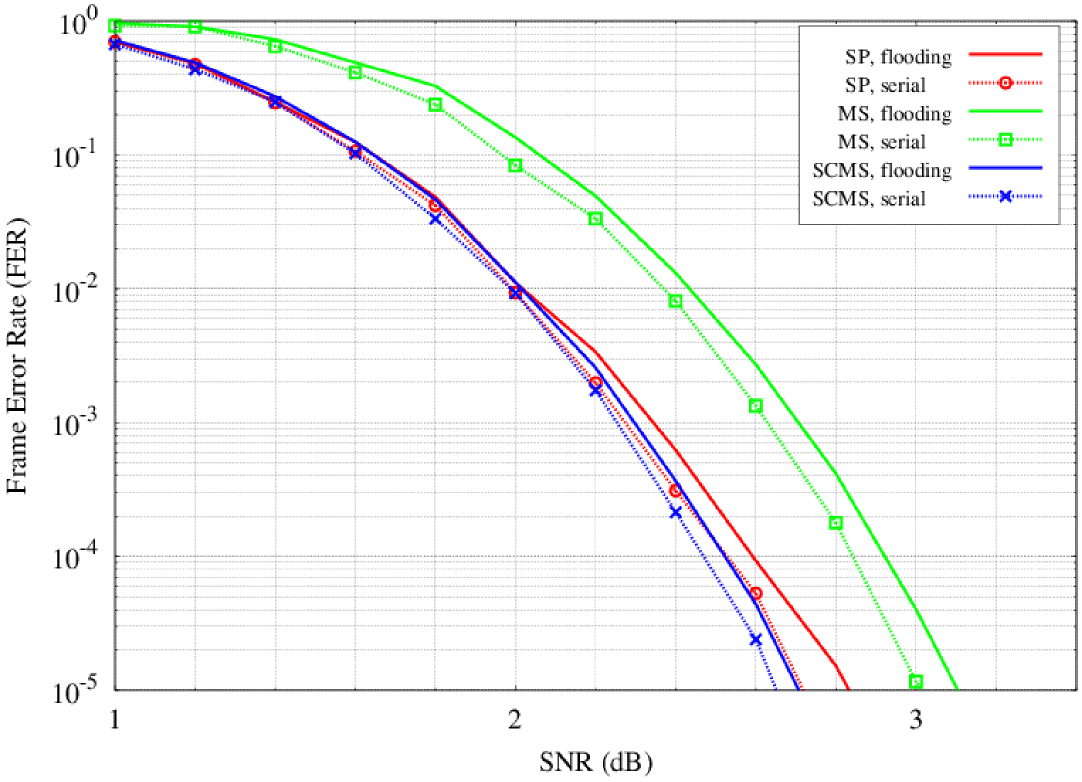
- Fixed-point decoders
  - AP-LLR ($\tilde{\gamma}_n$) quantized on **6 bits**
  - IN-LLR (γ_n) and exchanged messages ($\alpha_{m,n}$, $\beta_{m,n}$) quantized on **4 or 5 bits**

- Fixed-point arithmetic
  - **6-bit adders**
    - exchanged messages and IN/AP-LLR values are saturated if out of corresponding range
  - **noiseless** (“exact”) **arithmetic**

- all simulations: Mackay’s regular (3,6)-LDPC code, [K = 504, N = 1008]

Floating-point simulations

- AWGN, QPSK
- Mackay's regular (3,6)-LDPC code, [K = 504, N = 1008]
- Floating-point decoders



Decoder

Sum-Product

Min-Sum

Self-Corrected Min-Sum

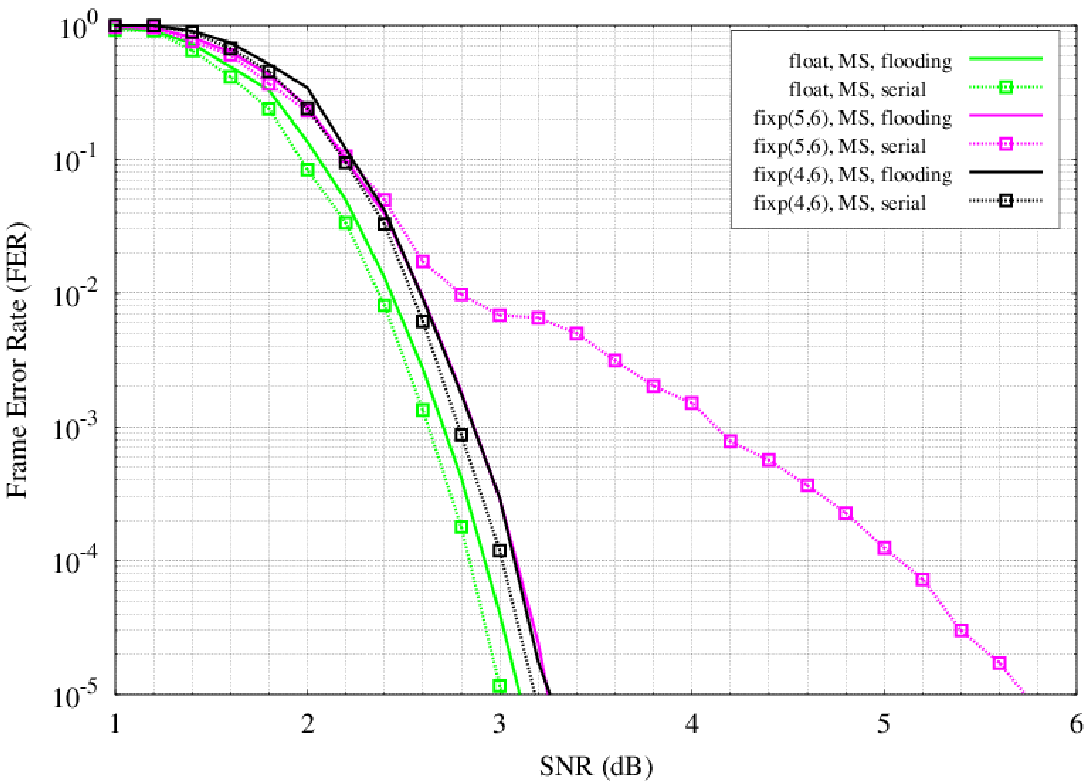
Scheduling

Solid curves: **flooding**

Dashed curves: **serial**

Fixed-point simulations / Min-Sum decoder

- AWGN, QPSK
- Mackay's regular (3,6)-LDPC code, [K = 504, N = 1008]
- Fixed-point quantization: (***x***, ***y***)
  - ***x*** bits for IN-LLR & exchanged messages, ***y*** bits for AP-LLR



Color code

floating-point

fixed-point(5, 6)

fixed-point(4, 6)

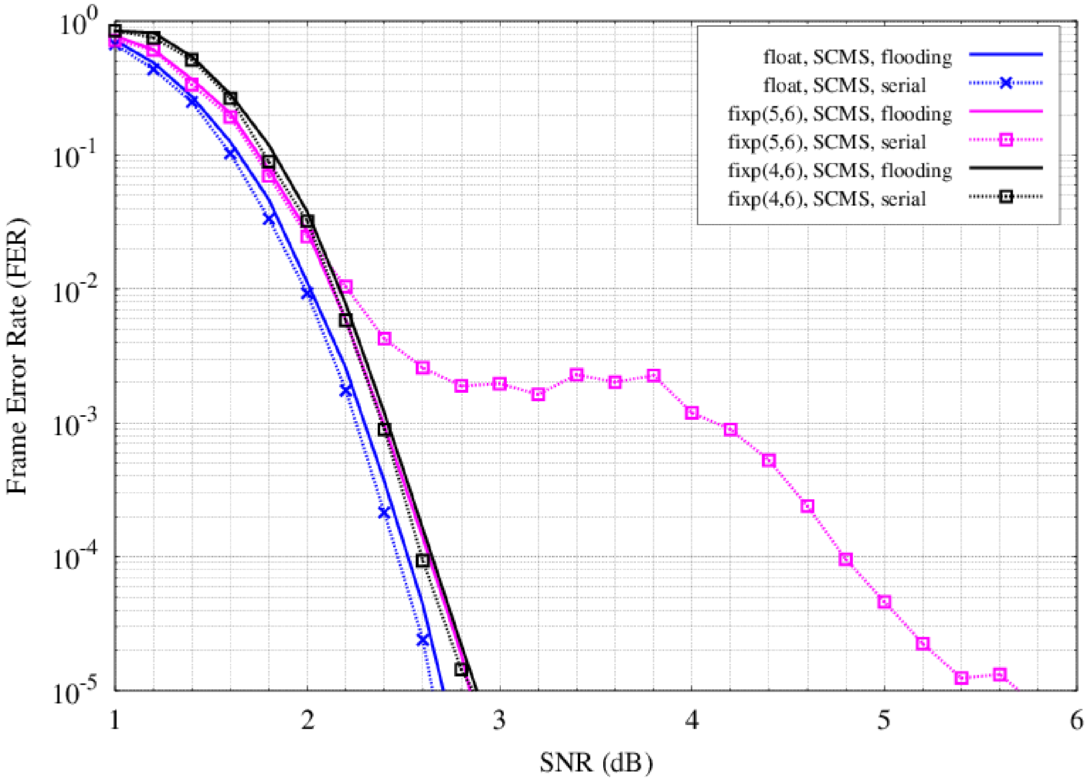
Scheduling

Solid curves: **flooding**

Dashed curves: **serial**

Fixed-point simulations / SC-Min-Sum decoder

- AWGN, QPSK
- Mackay's regular (3,6)-LDPC code, [K = 504, N = 1008]
- Fixed-point quantization: (***x***, ***y***)
  - ***x*** bits for IN-LLR & exchanged messages, ***y*** bits for AP-LLR



Color code

floating-point
fixed-point(5, 6)
fixed-point(4, 6)

Scheduling

Solid curves: **flooding**
Dashed curves: **serial**

Impact of the scheduling and of the quantization of the decoder performance
2. Noisy devices

- Fixed-point decoders
  - AP-LLR ($\tilde{\gamma}_n$) quantized on **6 bits**
  - IN-LLR (γ_n) and exchanged messages ($\alpha_{m,n}$, $\beta_{m,n}$) quantized on **4 bits**

- Fixed-point arithmetic
  - **6-bit adders**
    - exchanged messages and IN/AP-LLR values are saturated if out of corresponding range
  - **noisy** (probabilistic) **arithmetic**

- all simulations: Mackay’s regular (3,6)-LDPC code, [K = 504, N = 1008]

Min-Sum decoder on faulty devices

Initialization: $\forall n = 1, \dots, N;\ \forall m \in H(n)$

$$\gamma_n = \log(\Pr(x_n = 0 \mid y_n)/\Pr(x_n = 1 \mid y_n))$$
$$\alpha_{m,n} = \gamma_n$$

Iterations

- **CNU**: $\forall m = 1, \dots, M;\ \forall n \in H(m)$

$$\beta_{m,n} = \left(\prod_{n' \in H(m) \backslash n} \operatorname{sgn}(\alpha_{m,n'}) \right) \min_{n' \in H(m) \backslash n} (|\alpha_{m,n'}|)$$

comparators

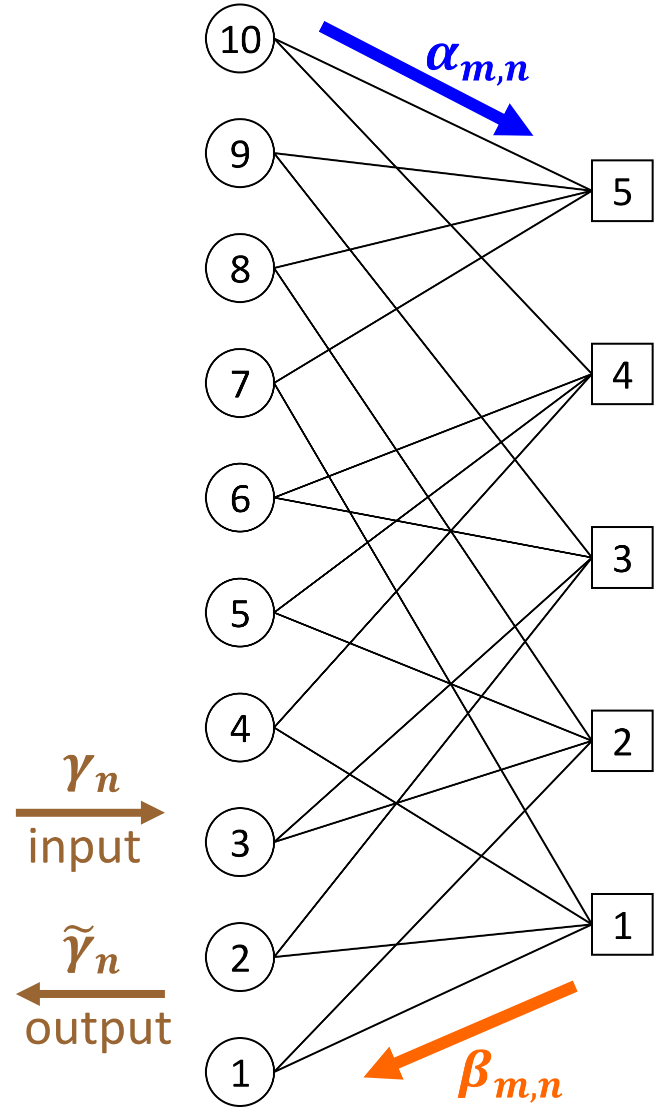
- **AP-LLR**: $\forall n = 1, \dots, N$

$$\tilde{\gamma}_n = \gamma_n + \sum_{m \in H(n)} \beta_{m,n}$$

adders

- **VNU**: $\forall n = 1, \dots, N;\ \forall m \in H(n)$

$$\alpha_{m,n} = \tilde{\gamma}_n - \beta_{m,n}$$



Error models for faulty components

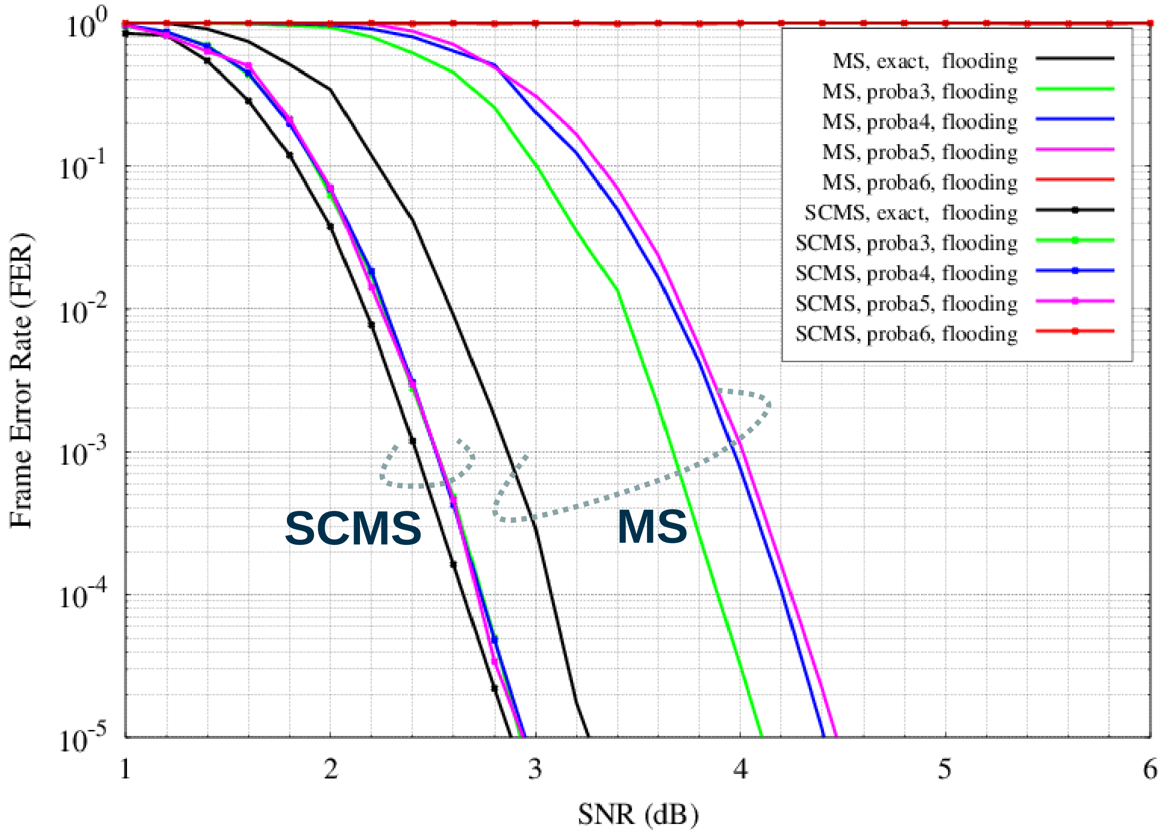
- **Probabilistic adder** (Q=6 bits)
  - Two parameters: the **depth *D*** and the **error probability P_a**
  - P_a is the probability that an error occurs on at least one of the D LSBs

	Q		D		2	1	
correct output	1	0	1	1	0	1	two's complement
error pattern			0	1	1	0	rand integer in [1, 2^D-1]
erroneous output	1	0	1	0	1	1	

- **Probabilistic comparator**
  - P_c is the probability that the output is in error

Flooding implementation / SCMS vs. MS

- AWGN, QPSK
- Mackay's regular (3,6)-LDPC code, [K = 504, N = 1008]
- Fixed-point decoders: 4 / 6 bits (IN-LLR & exchanged messages / AP-LLR)

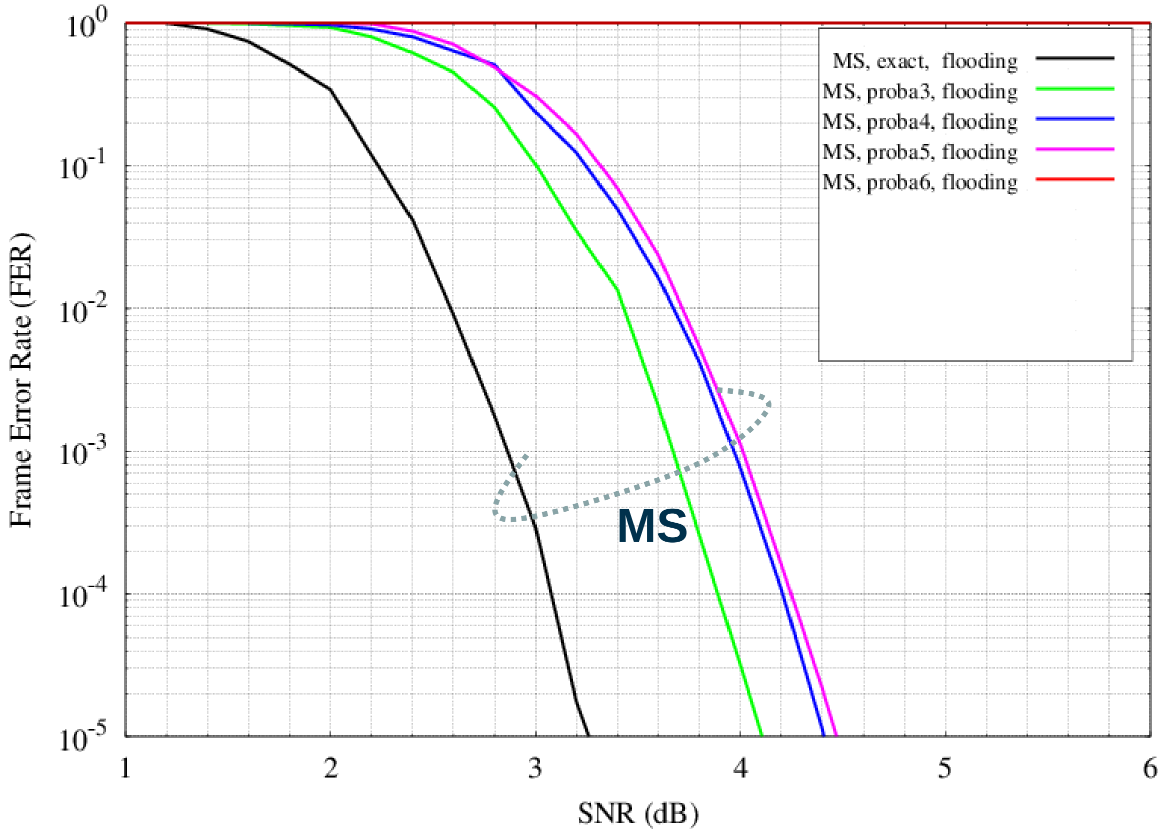


Comp.err. prob: $P_c = 0.01$
Adder err. prrob: $P_a = 0.01$

Color code:
Noiseless
Depth = 3
Depth = 4
Depth = 5
Depth = 6

MS decoder / flooding implementation

- AWGN, QPSK
- Mackay's regular (3,6)-LDPC code, [K = 504, N = 1008]
- Fixed-point decoders: 4 / 6 bits (IN-LLR & exchanged messages / AP-LLR)



Comp.err. prob: $P_c = 0.01$
Adder err. prrob: $P_a = 0.01$

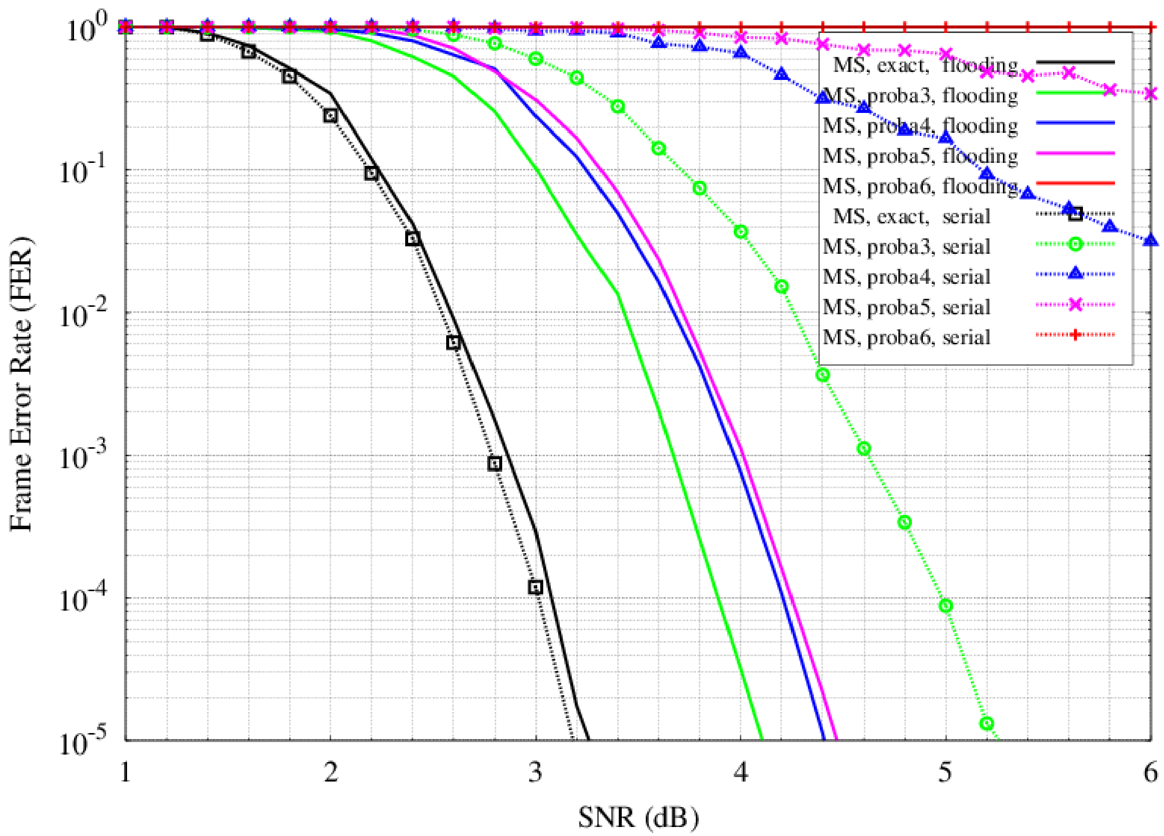
Color code:

Noiseless
Depth = 3
Depth = 4
Depth = 5
Depth = 6

Solid curves: **flooding**

MS decoder / flooding vs. serial implementation

- AWGN, QPSK
- Mackay's regular (3,6)-LDPC code, [K = 504, N = 1008]
- Fixed-point decoders: 4 / 6 bits (IN-LLR & exchanged messages / AP-LLR)



Comp.err. prob: $P_c = 0.01$
Adder err. prrob: $P_a = 0.01$

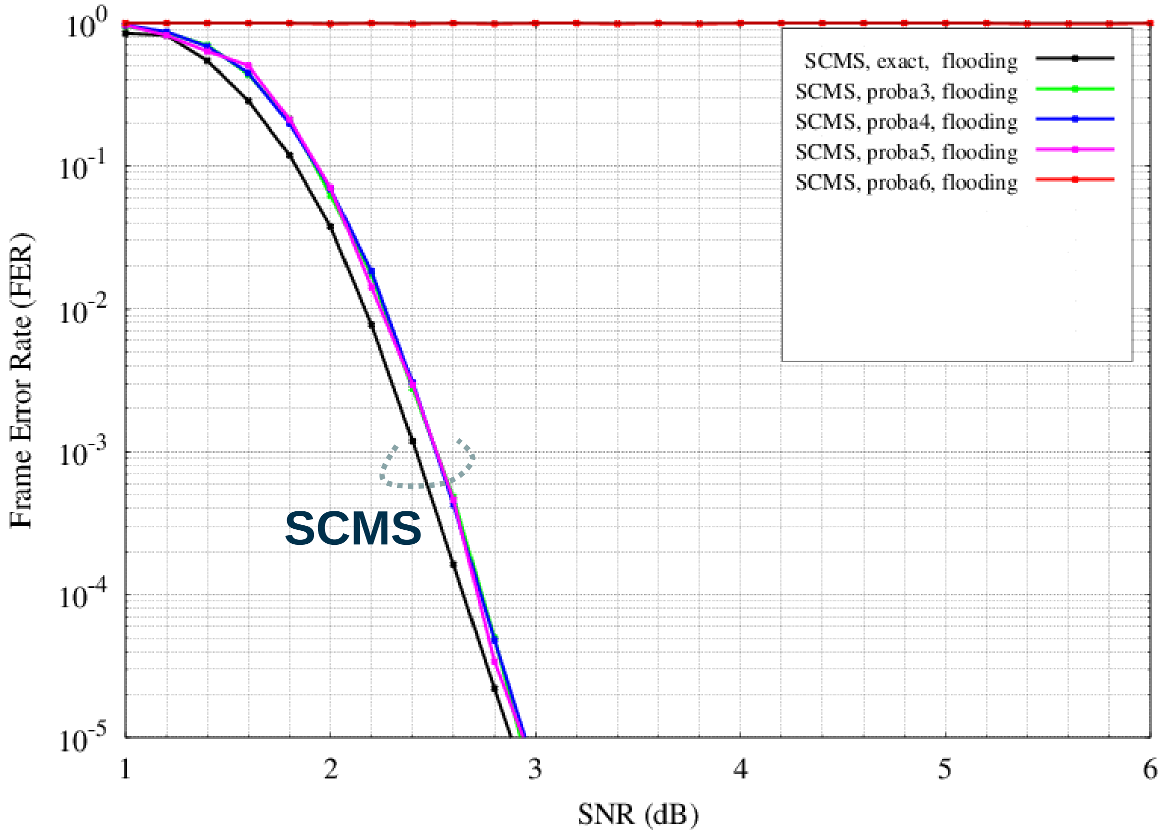
Color code:

Noiseless
Depth = 3
Depth = 4
Depth = 5
Depth = 6

Solid curves: **flooding**
Dashed curves: **serial**

SCMS decoder / flooding implementation

- AWGN, QPSK
- Mackay's regular (3,6)-LDPC code, [K = 504, N = 1008]
- Fixed-point decoders: 4 / 6 bits (IN-LLR & exchanged messages / AP-LLR)



Comp.err. prob: $P_c = 0.01$
Adder err. prrob: $P_a = 0.01$

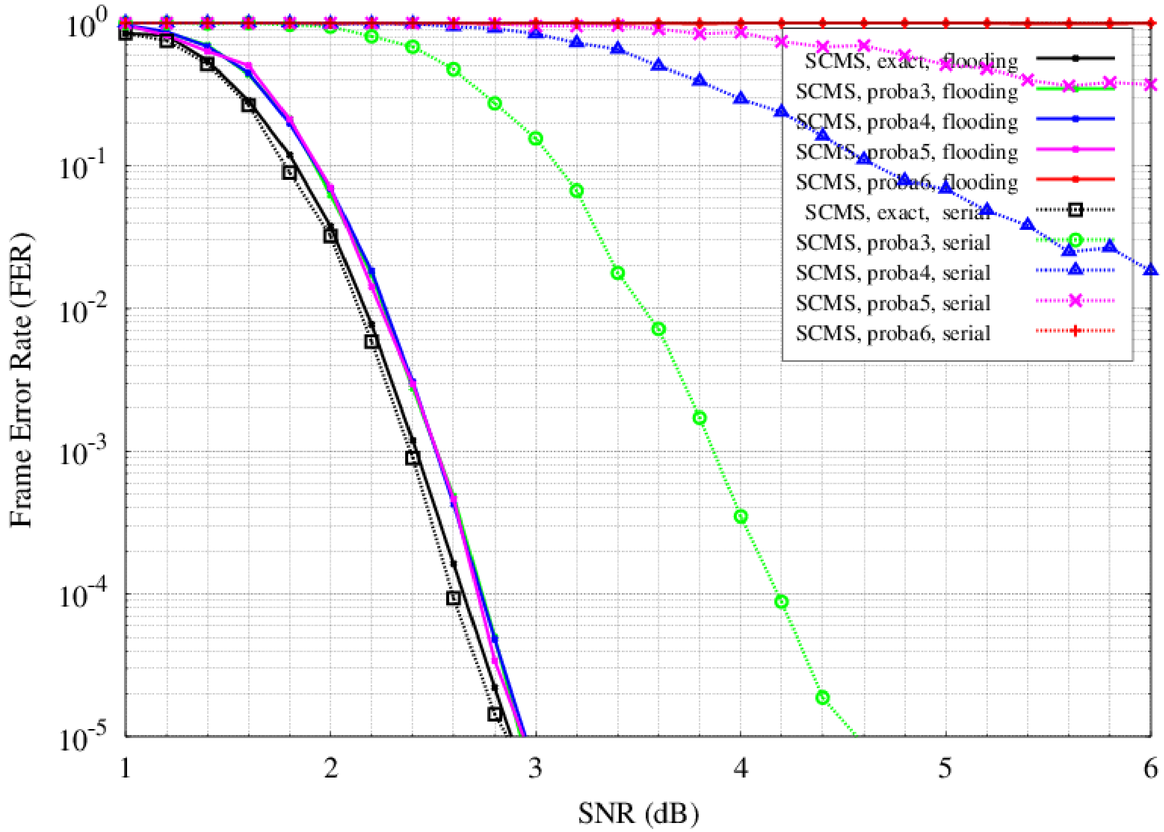
Color code:

Noiseless
Depth = 3
Depth = 4
Depth = 5
Depth = 6

Solid curves: **flooding**

SCMS decoder / flooding vs. serial implementation

- AWGN, QPSK
- Mackay's regular (3,6)-LDPC code, [K = 504, N = 1008]
- Fixed-point decoders: 4 / 6 bits (IN-LLR & exchanged messages / AP-LLR)



Comp.err. prob: $P_c = 0.01$

Adder err. prrob: $P_a = 0.01$

Color code:

Noiseless

Depth = 3

Depth = 4

Depth = 5

Depth = 6

Solid curves: **flooding**

Dashed curves: **serial**

Merci de votre attention

LABORATOIRE D'ÉLECTRONIQUE
ET DE TECHNOLOGIES
DE L'INFORMATION

CEA-Leti
MINATEC Campus, 17 rue des Martyrs
38054 GRENOBLE Cedex 9
Tel. +33 4 38 78 36 25

www.leti.fr