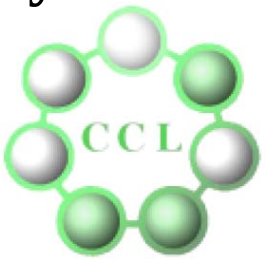
Performance Comparison of LDPC Convolutional Codes for Memory Size and Encoder Block Size



Yonsei Univ. Seoul, KOREA

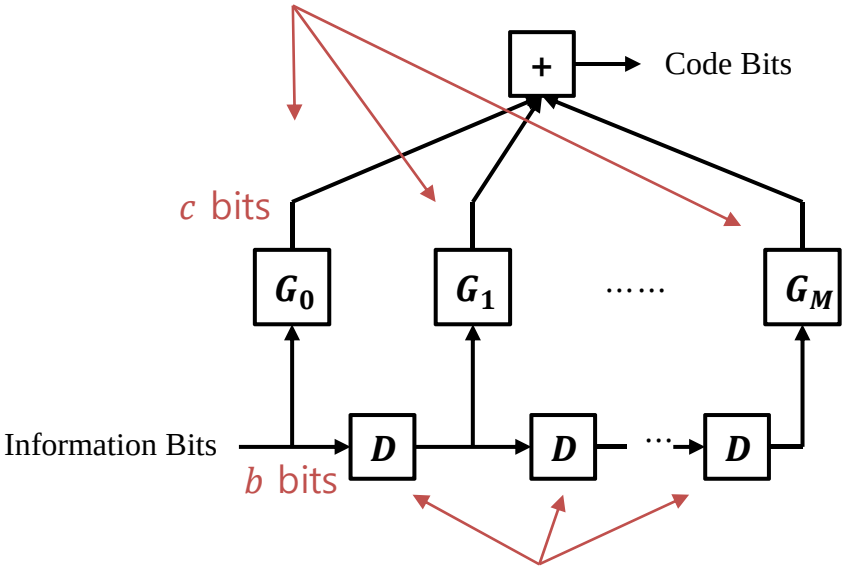
Ki-Hyeon Park, Jin Soo Park, Jung-Hyun Kim,

Inseon Kim, and Hong-Yeop Song

ICTC'13, Oct. 14 - 16, 2013

LDPC Convolutional Codes

Encoder Structure



Constraint Length = $b(M+1)$

Parity Check Matrix

$(c-b)\times c$

$$\boldsymbol{H}=\begin{bmatrix} \boldsymbol{H}_0 & & & \\ \boldsymbol{H}_1 & \boldsymbol{H}_0 & & \\ \vdots & \vdots & \ddots & \\ \boldsymbol{H}_M & \boldsymbol{H}_{M-1} & & \\ & \boldsymbol{H}_M & & \\ & & \ddots & \boldsymbol{H}_0 \\ & & & \vdots \\ & & & \boldsymbol{H}_{M-1} \\ & & & \boldsymbol{H}_M \end{bmatrix} \} M+1$$

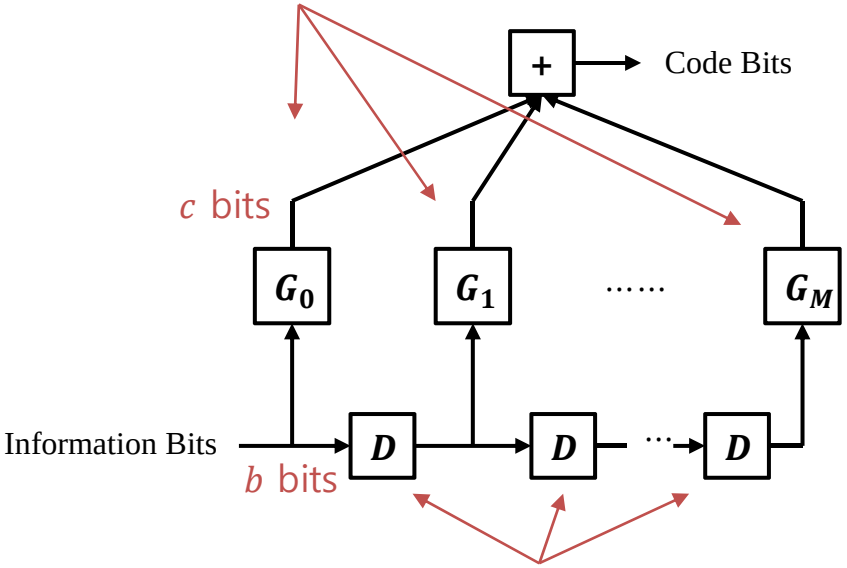
N Columns

$(c-b)(N+M)\times cN$

Parameters and Performance

Encoder Structure

$M+1$ **Generator** Matrices



M **Memories** (Delay Blocks)

Constraint Length = $b(M+1)$

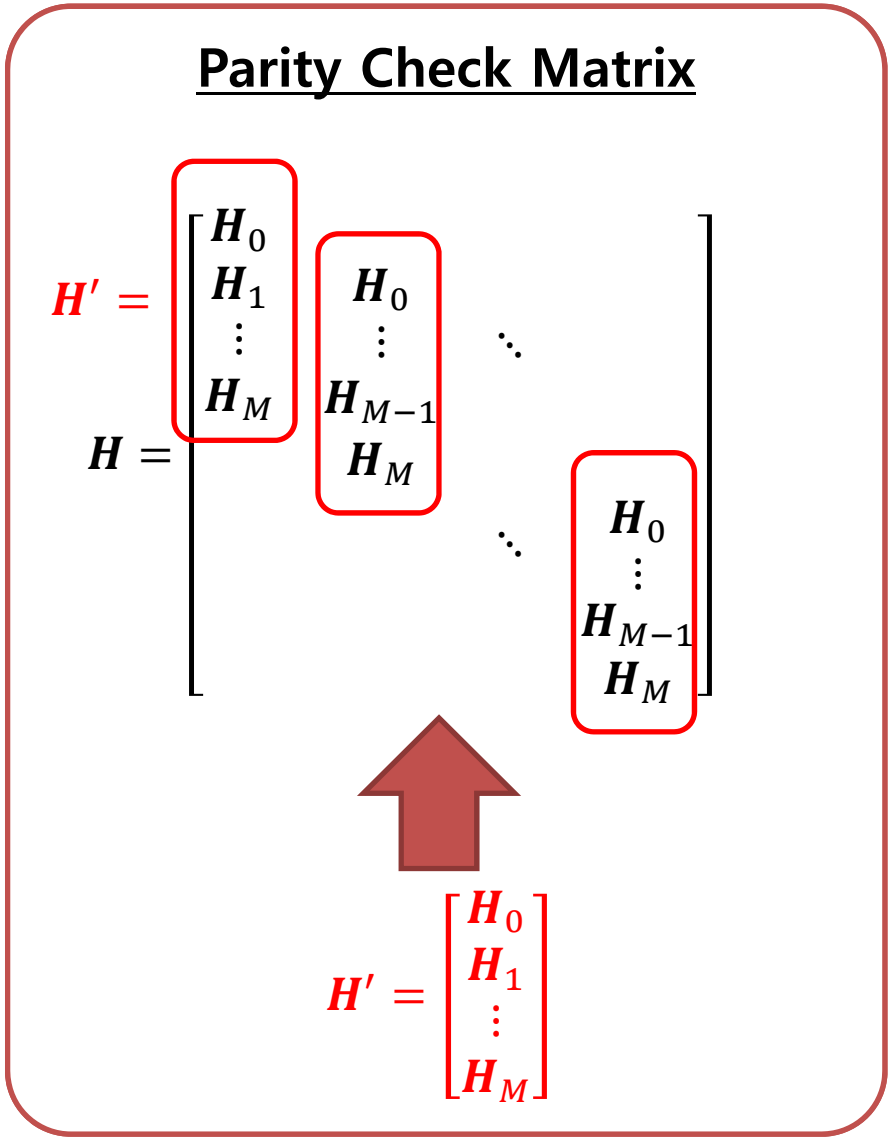
Known:

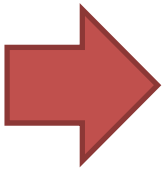
More Memories
→ Better Performance

Unknown:

Large/Small Size of Generator (equivalently, Parity-Check) Matrices
→ Better Performance?

Design of the Parity Check Matrix





Design H'
for
Better H

Design of the Sub-Matrices

There are some methods…
Idea from Quasi-Cyclic LDPC (**QC-LDPC**) block codes has been considered

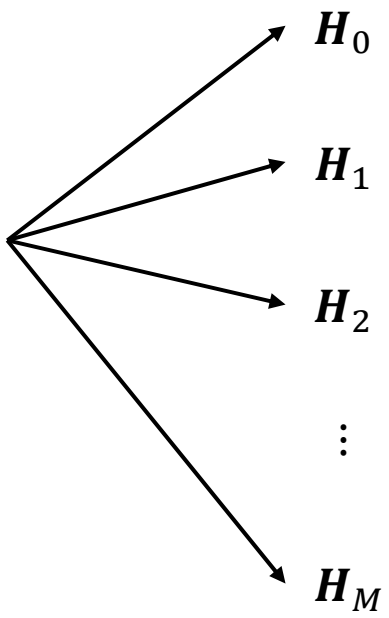
Design a Good $\boldsymbol{H(D)}$

$$\boldsymbol{H}(D) = \begin{bmatrix} h_{1,1}(D) & \cdots & h_{1,c}(D) \\ \vdots & \ddots & \vdots \\ h_{c-b,1}(D) & \cdots & h_{c-b,c}(D) \end{bmatrix}$$, where $h_{i,j}(D) = D^q, q = 0, \cdots, M.$

$(c-b) \times c$ monomial matrix

From $\boldsymbol{H}(D)$ to $\boldsymbol{H}_i$

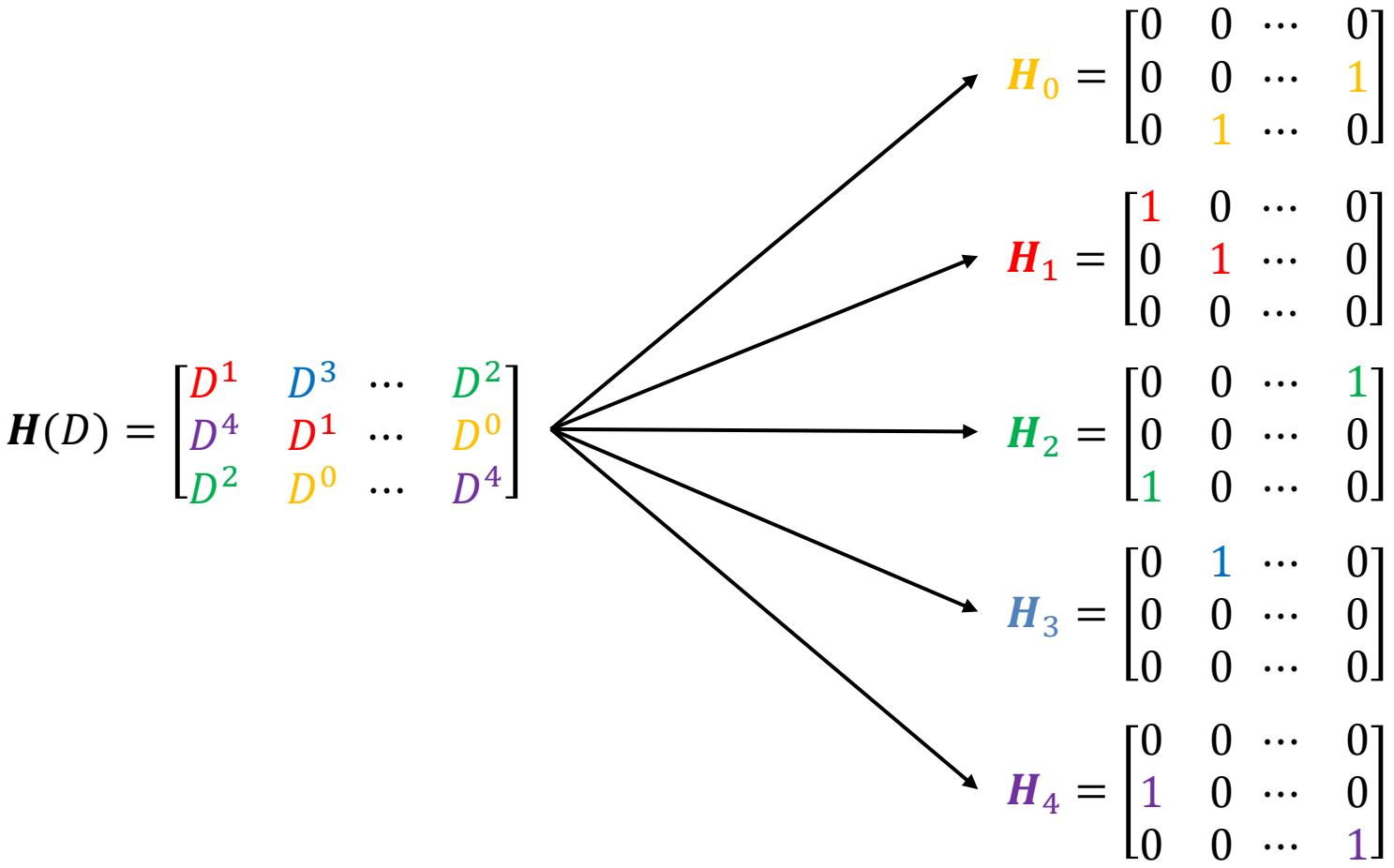
$$\boldsymbol{H}(D) = \begin{bmatrix} h_{1,1}(D) & \cdots & h_{1,c}(D) \\ \vdots & \ddots & \vdots \\ h_{c-b,1}(D) & \cdots & h_{c-b,c}(D) \end{bmatrix}$$



Position of 1 in $\boldsymbol{H}_i$ = Position of D^i in $\boldsymbol{H}(D)$
All other positions in $\boldsymbol{H}_i$: Put 0

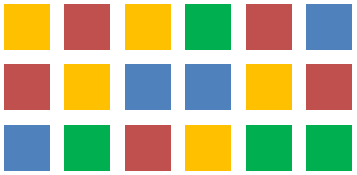
From $H(D)$ to H_i

Example

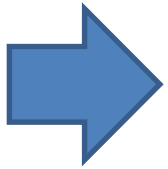


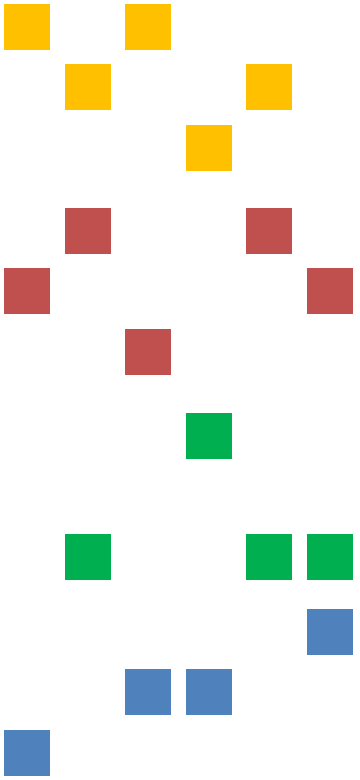
Matrix Expansion and Degree Distribution

Example: Comparing 3×6 and 6×12 monomial matrix

$\boldsymbol{H_{3\times6}(D)} =$ 

3 elements in a column



$\boldsymbol{H'} =$ 

3 elements in a column, too:
Degree of variable nodes = 3

Matrix Expansion and Degree Distribution

Example: Comparing 3×6 and 6×12 monomial matrix

What we want

Performance variation

by

Size of G & H

What we DON'T want

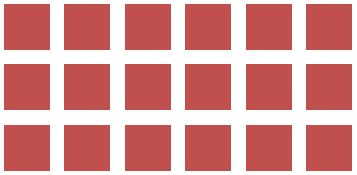
Performance variation

by

Degree distribution

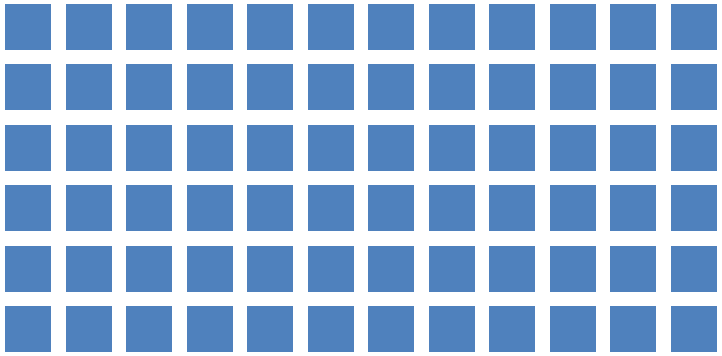
Matrix Expansion and Degree Distribution

Example: Comparing 3×6 and 6×12 monomial matrix

$H_{3\times6}(D) =$ 

Degree of variable nodes = 3

3 elements in a column

$H_{6\times12}(D) =$ 

Degree of variable nodes = 6

6 elements in a column

Sparse Monomial Matrix

Sparse Monomial Matrix

$$\boldsymbol{H}(D) = \begin{bmatrix} h_{1,1}(D) & \cdots & h_{1,c}(D) \\ \vdots & \ddots & \vdots \\ h_{c-b,1}(D) & \cdots & h_{c-b,c}(D) \end{bmatrix}$$, where $h_{i,j}(D) = \begin{cases} D^q, & q = 0, \cdots, M \\ 0 \end{cases}$.

Contains *a lot of zeros*

To change (or increase) the encoder block size
without changing the degree distribution of $\mathbf{H}'$

Matrix Expansion Operation

Definition (Matrix expansion operation): *Let* $\boldsymbol{A} = (\boldsymbol{a_{ij}})$ *be an* $m \times n$ *matrix and* $\boldsymbol{B} = (\boldsymbol{b_{ij}})$ *an* $ml \times nl$ $(0,1)$ *matrix for some integer* $l > 1$. *We define an operation* $\circledast$ *as*

$$\boldsymbol{A} \circledast \boldsymbol{B} = \boldsymbol{C},$$

where the $ml \times nl$ *matrix* $\boldsymbol{C} = (c_{ij})$ *is calculated as*

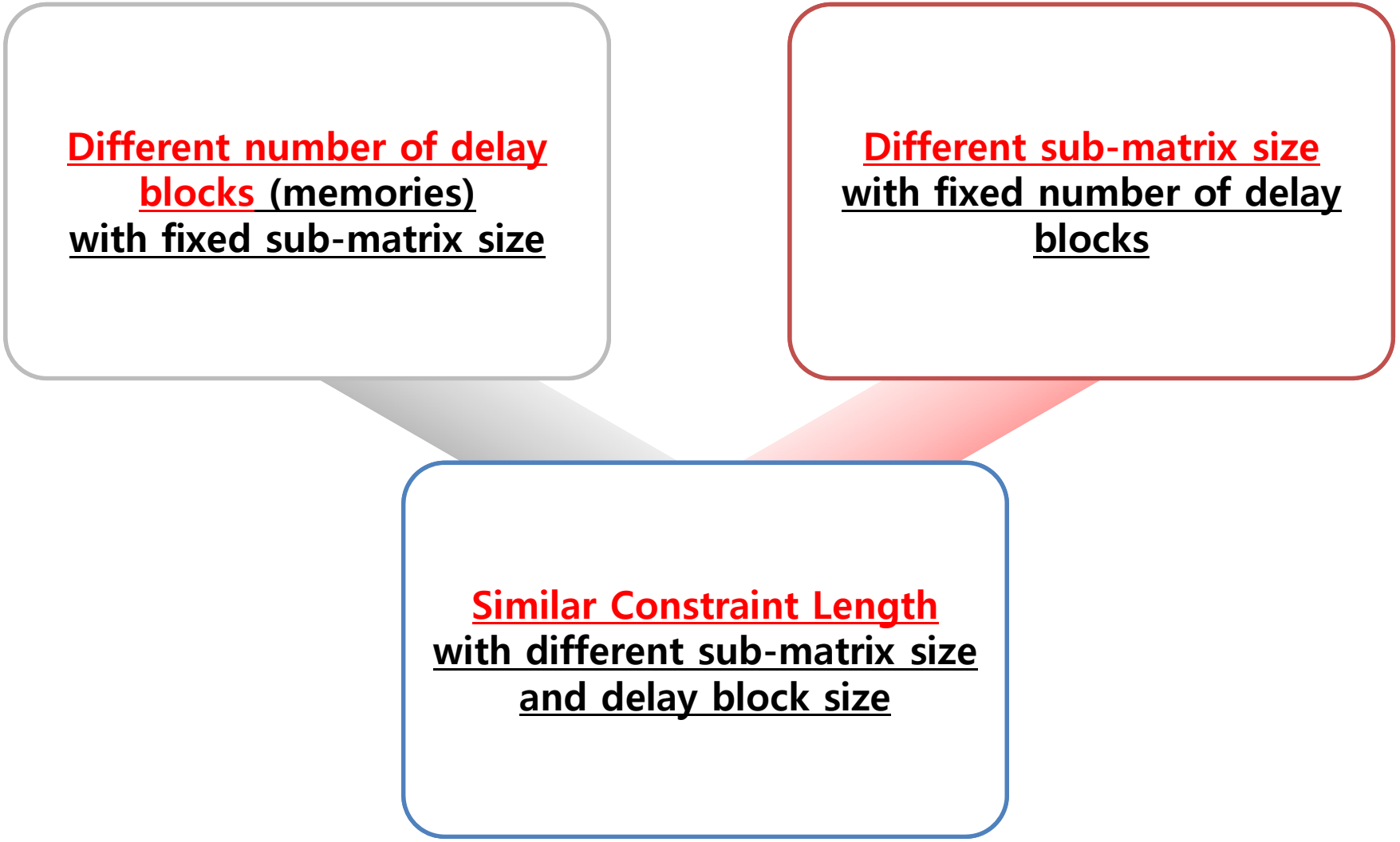
$$c_{ij} = b_{ij} \cdot a_{\left\lfloor \frac{i-1}{l} \right\rfloor + 1, \left\lfloor \frac{j-1}{l} \right\rfloor + 1}.$$

Example

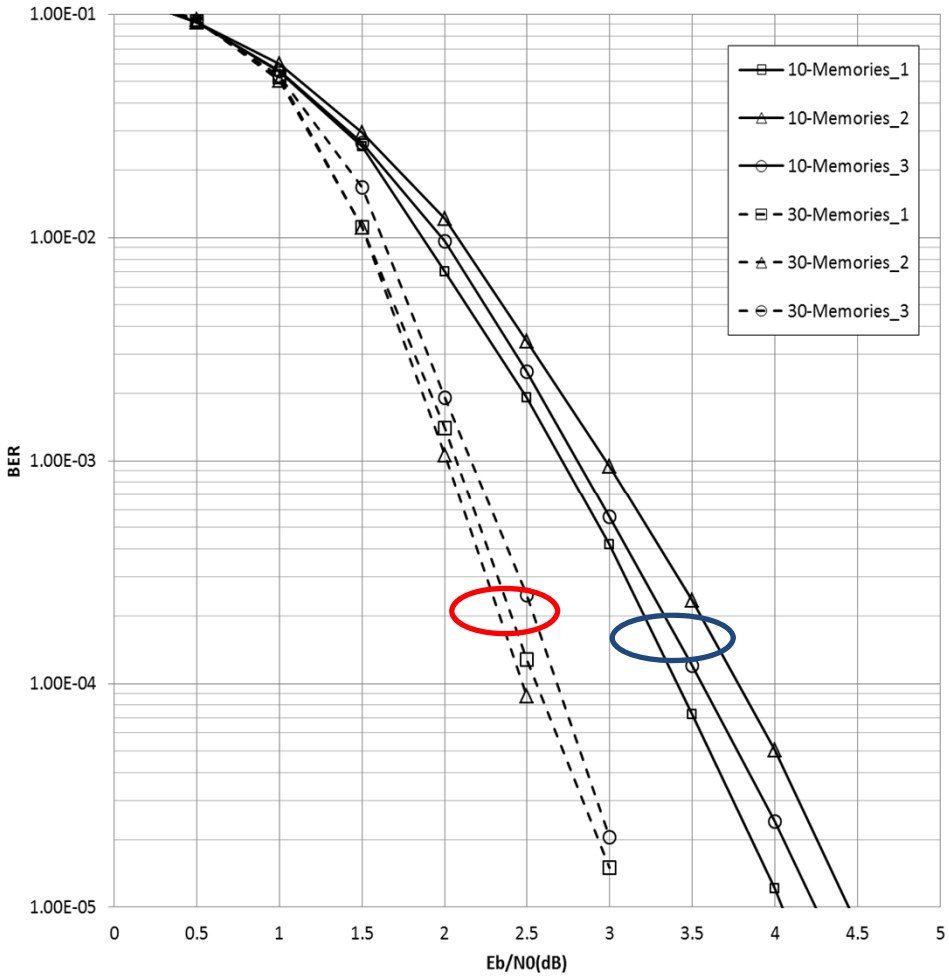
$$\begin{bmatrix} \boldsymbol{D^0} & \boldsymbol{D^1} \\ \boldsymbol{D^1} & \boldsymbol{D^3} \end{bmatrix} \circledast \left[\begin{array}{cc|cc} \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{1} \\ \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} \end{array}\right] = \left[\begin{array}{cc|cc} \boldsymbol{D^0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{D^1} \\ \boldsymbol{0} & \boldsymbol{D^0} & \boldsymbol{D^1} & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{D^1} & \boldsymbol{0} & \boldsymbol{D^3} \\ \boldsymbol{D^1} & \boldsymbol{0} & \boldsymbol{D^3} & \boldsymbol{0} \end{array}\right]$$

$\boldsymbol{A}$	$\boldsymbol{B}$	$\boldsymbol{C}$
Target monomial matrix	Expansion pattern (QC-LDPC form)	Expanded matrix

Control Parameters for Simulation



Simulation: Different Number of Delay Blocks

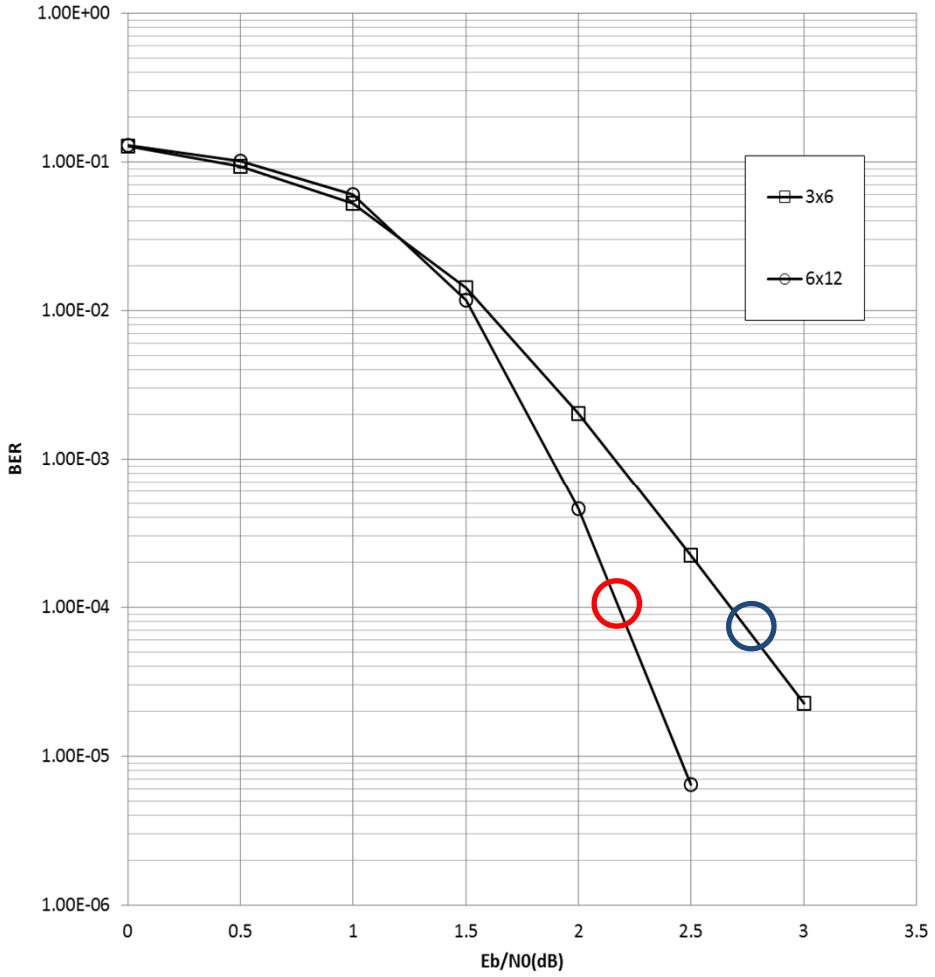


Same sub-matrix size
Different delay block size

Code Parameters				
n	***n-k***	***Sub-matrix size***	***Delay block size***	***Code rate***
1800	930	**3×6**	**10**	0.48
	990		**30**	0.45

More delay blocks
→ Better performance
(well known results)

Simulation: Different Sub-Matrix Size

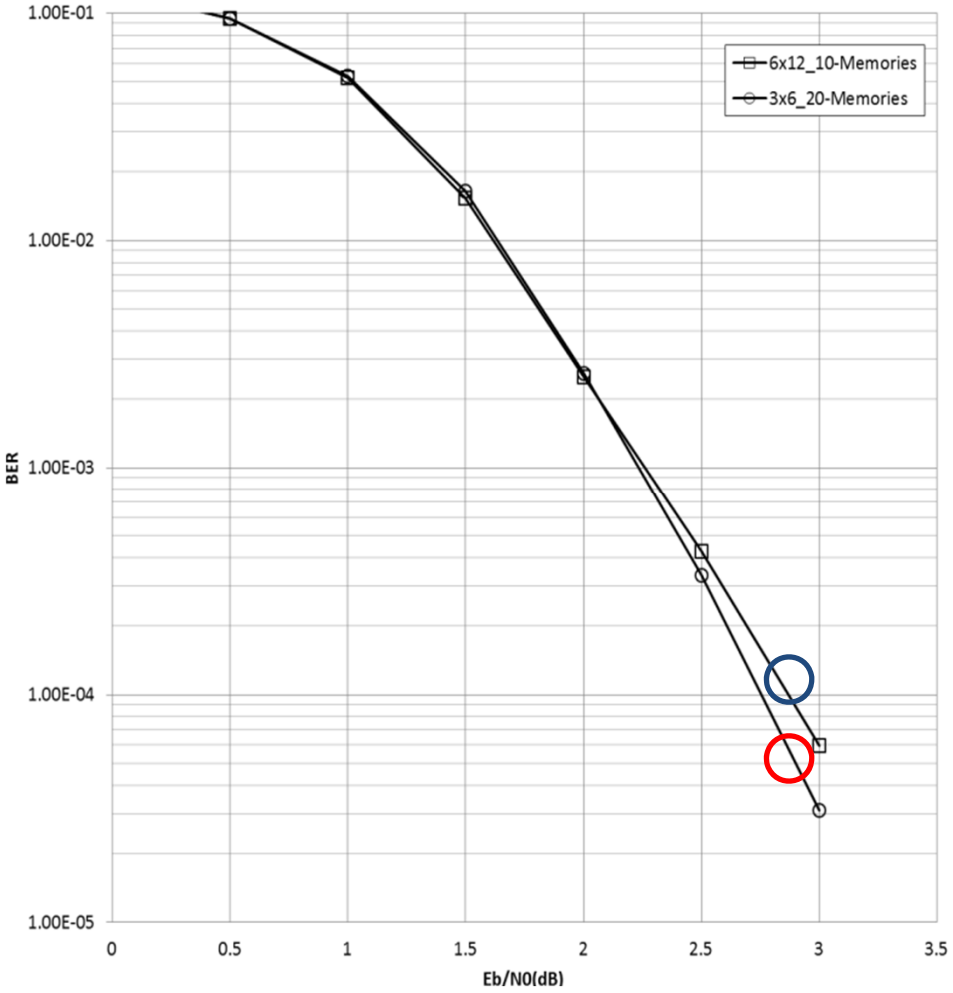


Different sub-matrix size
Same delay block size

Code Parameters				
n	***n-k***	***Sub-matrix size***	***Delay block size***	***Code rate***
1800	960	**3×6**	**20**	0.47
	1020	**6×12**		0.43

Large sub-matrix size
→ Better performance

Simulation: Similar Constraint Length



Similar constraint length
Different sub-matrix size and delay block size

Code Parameters					
n	***n-k***	***Sub-matrix size***	***Delay block size***	***Constraint length***	***Code rate***
1800	960	3×6	20	63	0.47
		6×12	10	66	

Similar performance

Discussion

Number of Delay Blocks	• Larger is Better • Matched with convolutional codes
Size of Sub-Matrix Size	• Bigger is Better • Relaxed condition for optimization (but we do not consider in this paper)
Similar Constraint Length	• Similar performance • Conjecture: similar constraint length gives similar performance