Fast List Decoding of Polar Codes: Algorithms and Implementation

Sravan Kumar Ankireddy

Guide: *Dr. Andrew Thangaraj* and *Dr. Radha Krishna Ganti*
Department of Electrical Engineering
IIT Madras

B-DMC Channel

- Let $W : X \longrightarrow Y$ be a Binary-Input Discrete Memoryless Channel as shown:

  where input alphabet: $\mathcal{X}$ = {0,1} and output alphabet: $\mathcal{Y}$

- The transition probabilities for the channel are given by: $W(y|x)$

  where x $\in$ $\mathcal{X}$ and y $\in$ $\mathcal{Y}$

- Assume that the channel has input-output symmetry. ***Ex***: BEC, BSC

Symmetric Channel Capacity

- Symmetric channel capacity is used as a "*measure of goodness*" for the channels
- For the B-DMC channel described above, capacity is given by

$$I(\mathcal{W}) = \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} \frac{1}{2} \mathcal{W}(y|x) \log_2 \frac{\mathcal{W}(y|x)}{\frac{1}{2}\mathcal{W}(y|0) + \frac{1}{2}\mathcal{W}(y|1)}$$

where

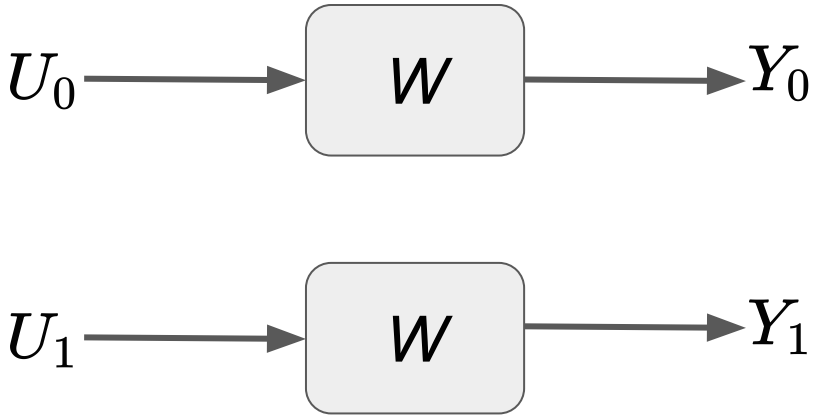
$$\mathcal{X} \in \{0, 1\}$$

- The two trivial/extreme channels are:
  - Perfect Chanel, highly reliable: *I(W)* = 1
  - Useless Channel, completely unreliable: *I(W)* = 0

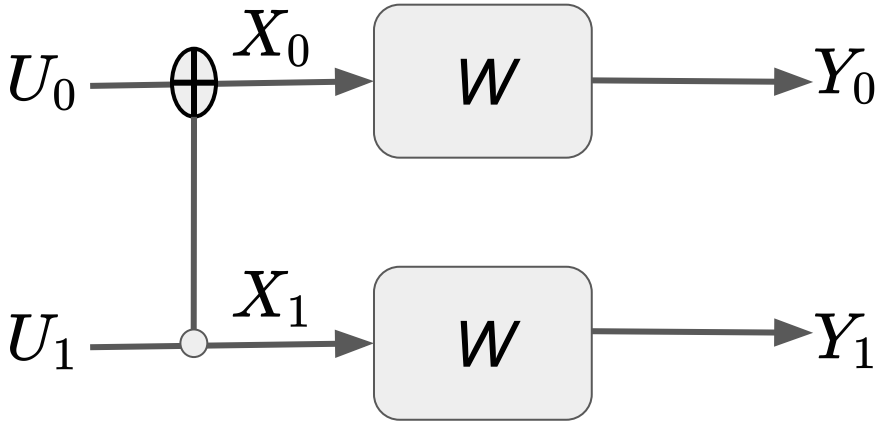
Channel Polarization

- Transform the original set of channels into extreme channels (good or bad)
- The total capacity remains constant and just gets redistributed
- As the number of transformed channels increases, the probability of error for estimating each symbol approaches either 1 or 0.5
- As the number of transformed channels increases, fraction of reliable bits approaches the channel capacity

Channel Polarization



$$I(Y_0, Y_1; U_0) = I(W) = I(Y_0, Y_1; U_1)$$



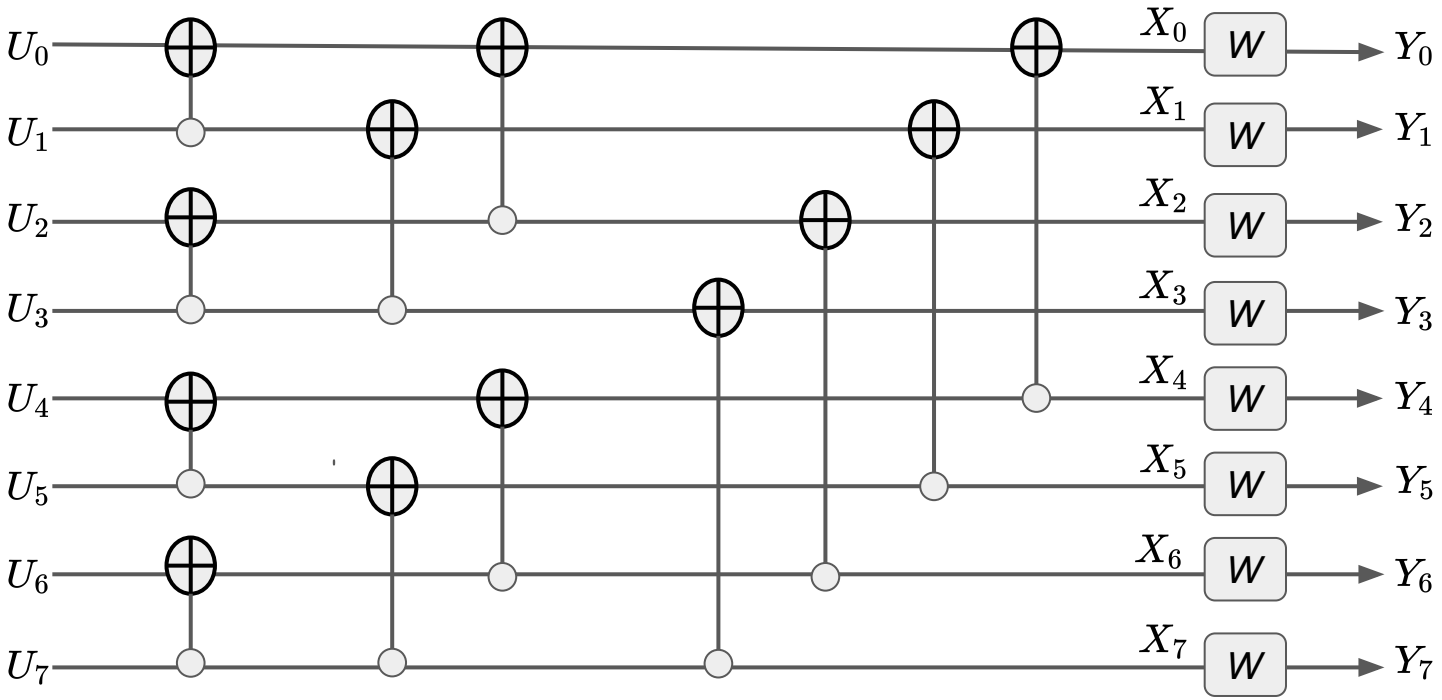
$$I(Y_0, Y_1; U_0) \leq I(W) \leq I(Y_0, Y_1; U_1)$$

New Channels: $W_0 : U_0 \to (Y_0, Y_1)$

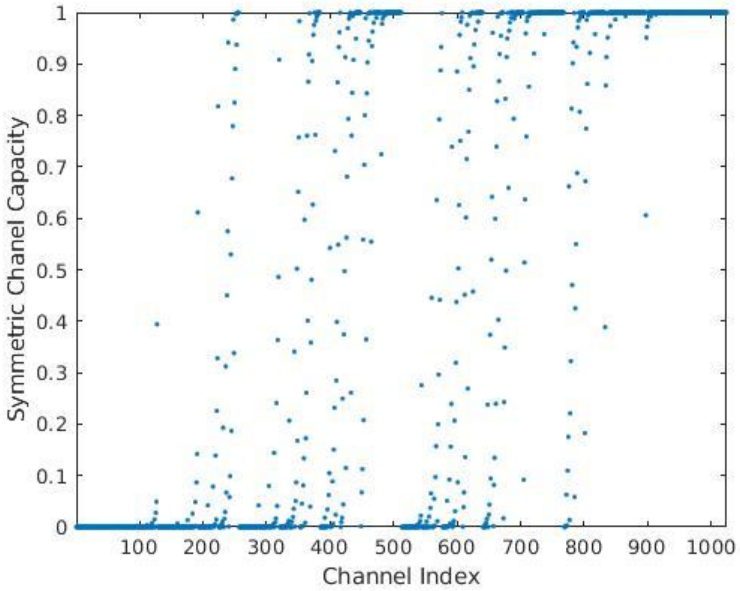
$W_1 : U_1 \to (Y_0, Y_1, U_0)$

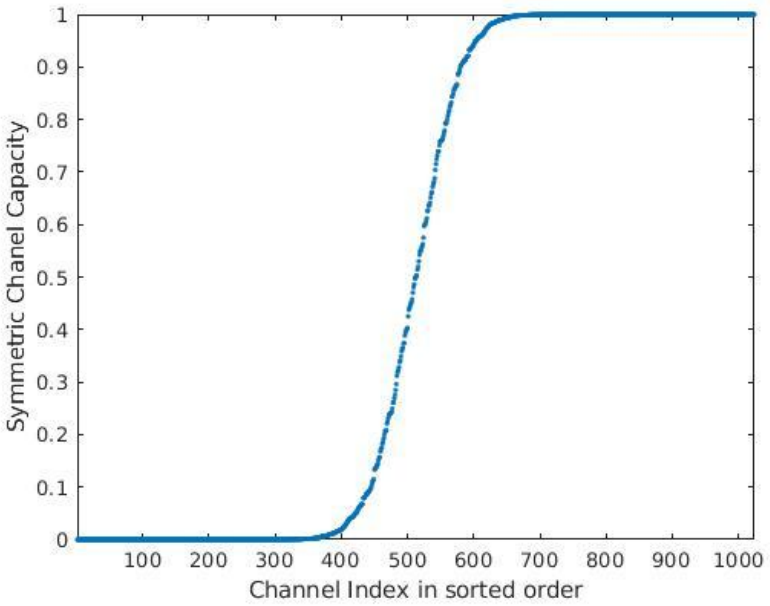
Channel Polarization: N = 8 polar code

x

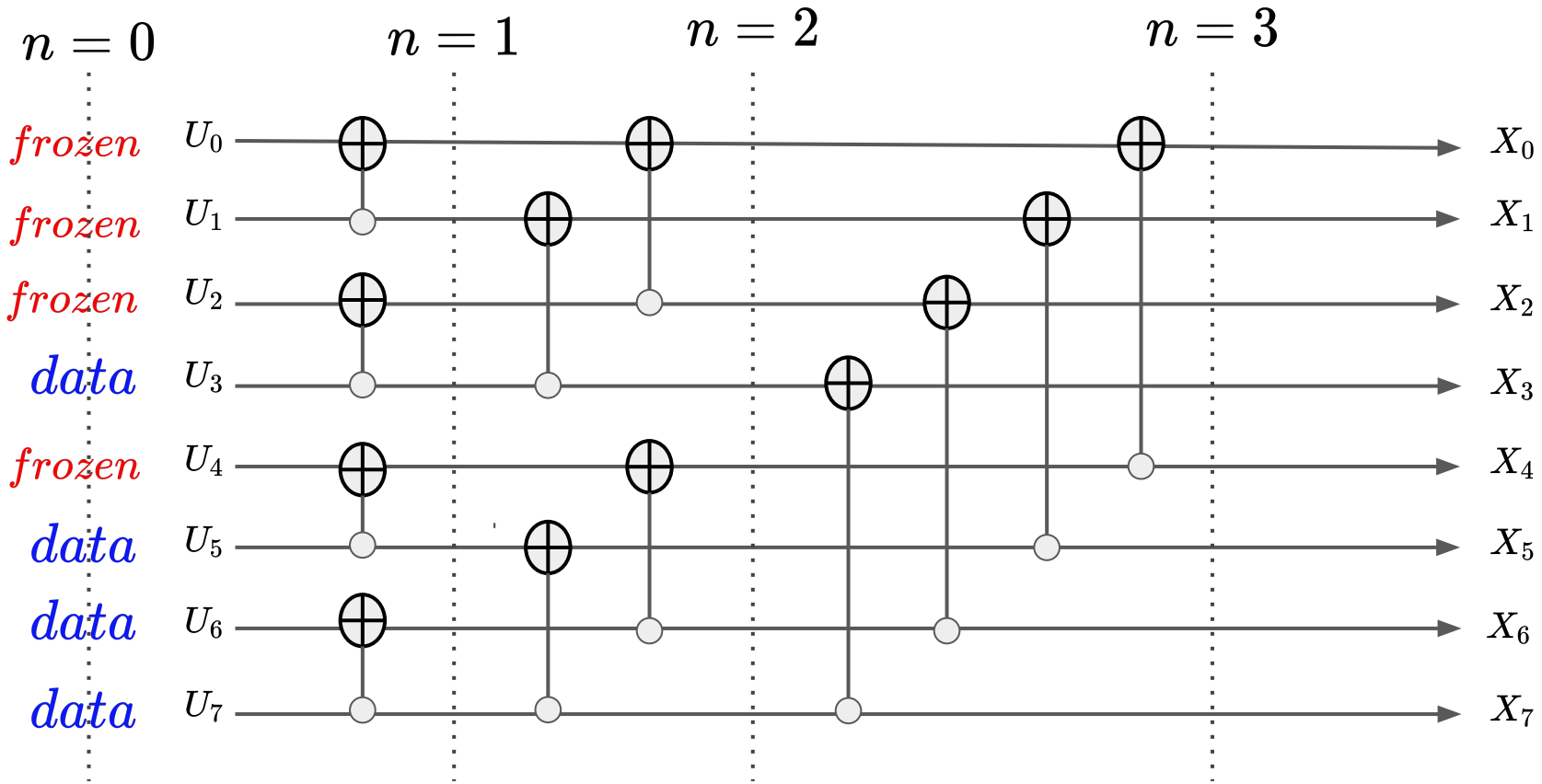


Polarization Effect: $BEC(\frac{1}{2})$, *N = 1024*

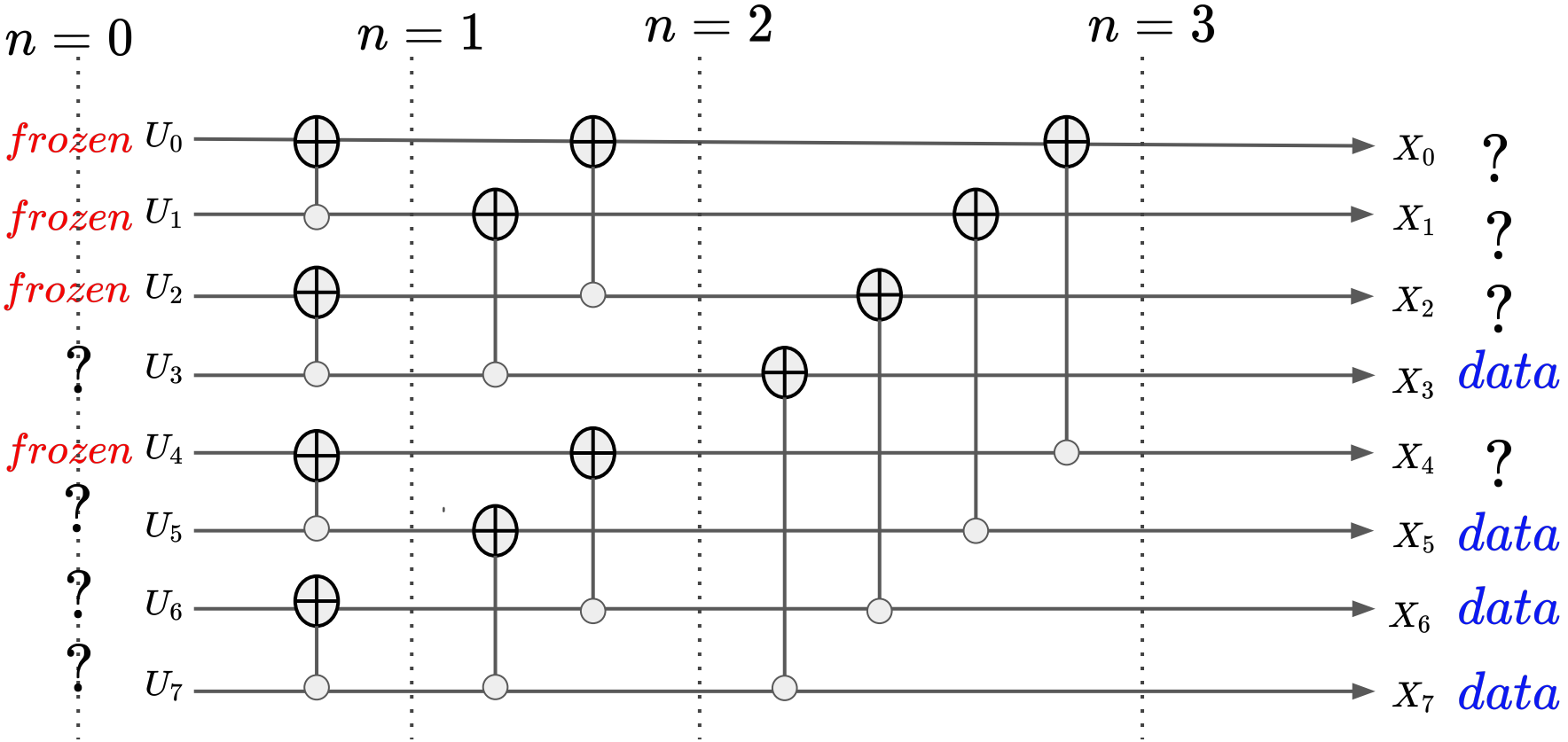




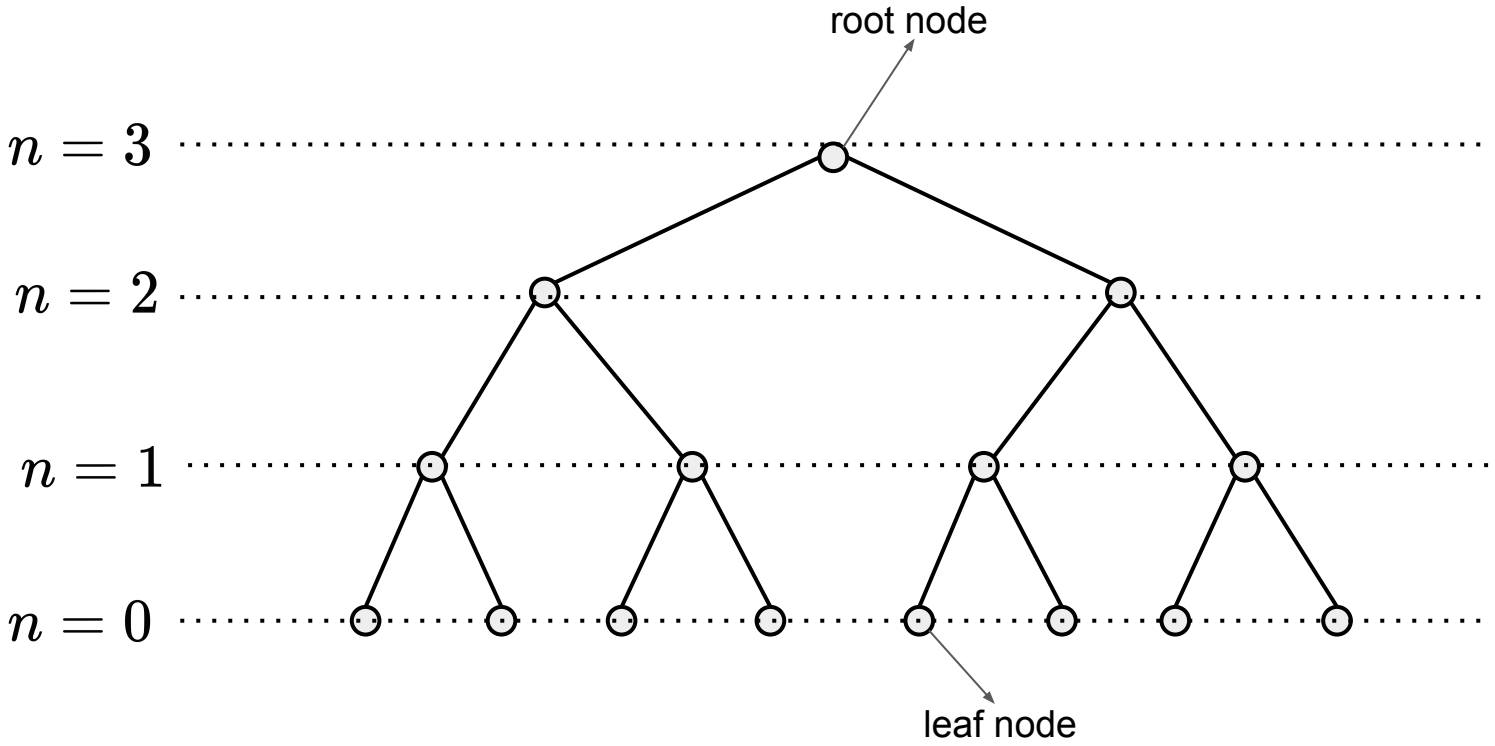
Non-systematic Encoding: $BEC(\frac{1}{2})$, (8,4) Polar Code



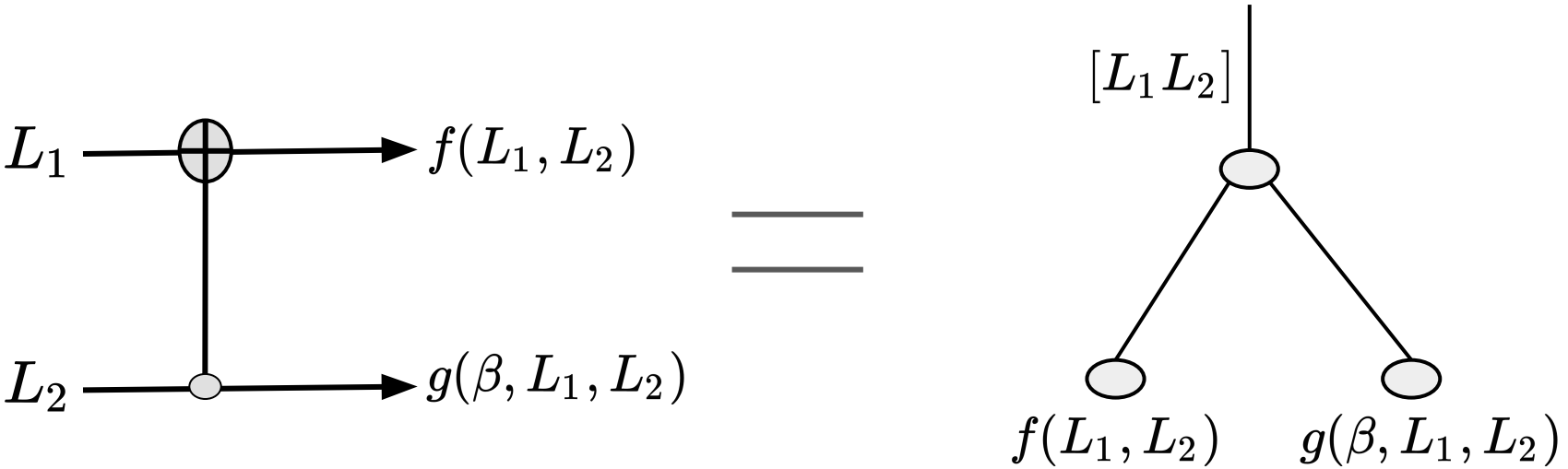
Systematic Encoding: $BEC(\frac{1}{2})$, (8,4) Polar Code



Binary Tree Representation of Polar Code



Processing Unit



- *L1* and *L2* are vectors; *f* and *g* are vector functions
- α_l = *f* ; α_r = *g*

Exact and Hardware Friendly versions

Exact: $f(L_1, L_2) = ln(\frac{1+e^{L_1+L_2}}{e^{L_1}+e^{L_2}})$

$$g(L_1, L_2) = (1-2\beta)L_1 + L_2$$

- The f function is computationally expensive and introduces roundoff errors
- Hence f is replaced with a Hardware Friendly (HWF), version which is the min-sum approximation, as follows:

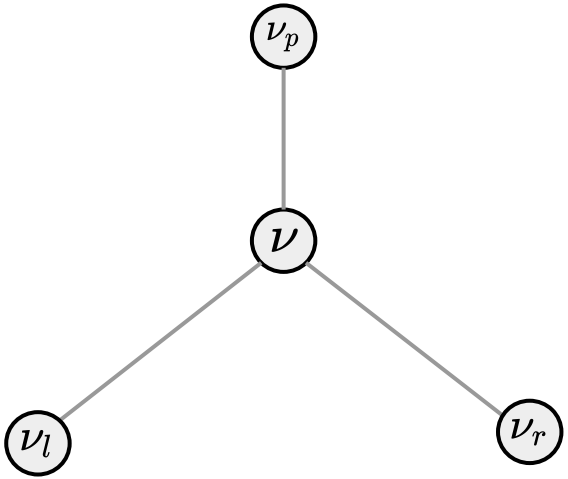
HWF: $f(L_1, L_2) = sign(L_1)sign(L_2)min(|L_1|, |L_2|)$

Successive Cancellation (SC) Decoding : algorithm

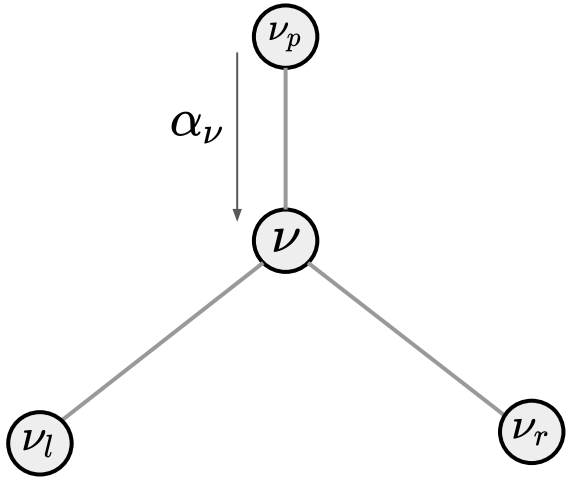
- Starting at the root node, traverse the binary tree in-order and at each node perform one of the following operations:
  - If current node has a left child that was not visited, calculate α_l and move to left child
    - This operation continues till you reach the first leaf node
  - If current node has a right child that was not visited, calculate α_r and move to right child
  - If both the messages from child nodes are available, calculate β and move to parent node
- At leaf node, hard decision is made according to the binary quantizer function as:

$$\beta_\nu = h(\alpha_\nu) = \begin{cases} 0, & \text{if } \alpha_\nu \geq 0 \\ 1, & \text{else} \end{cases}$$

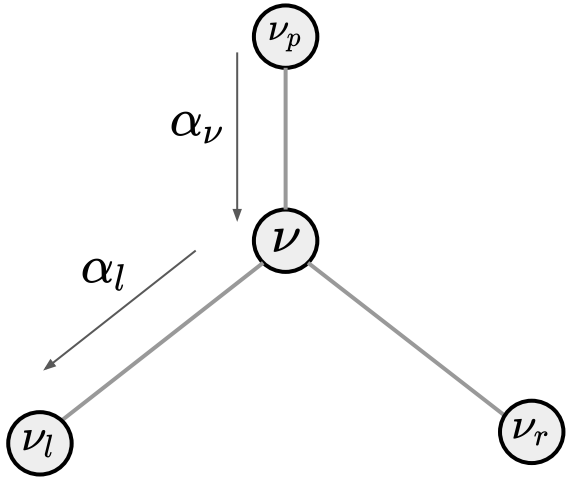
Successive Cancellation Decoding: Processing element



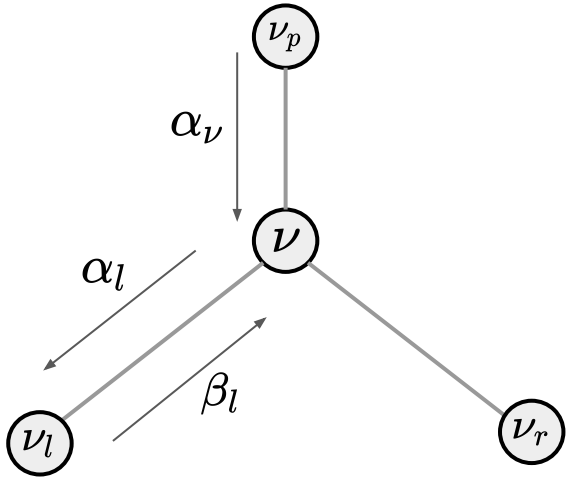
Successive Cancellation Decoding: Processing element



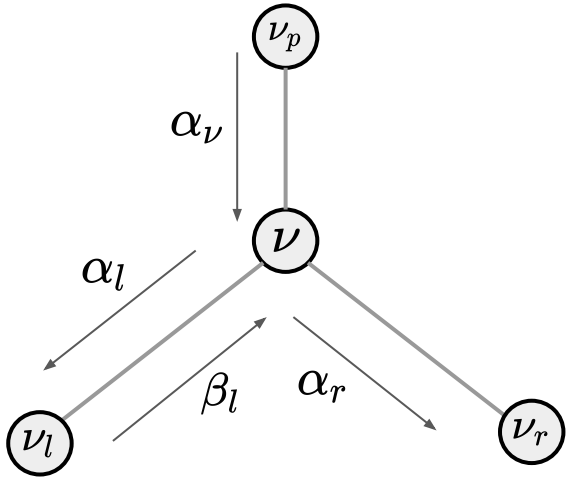
Successive Cancellation Decoding: Processing element



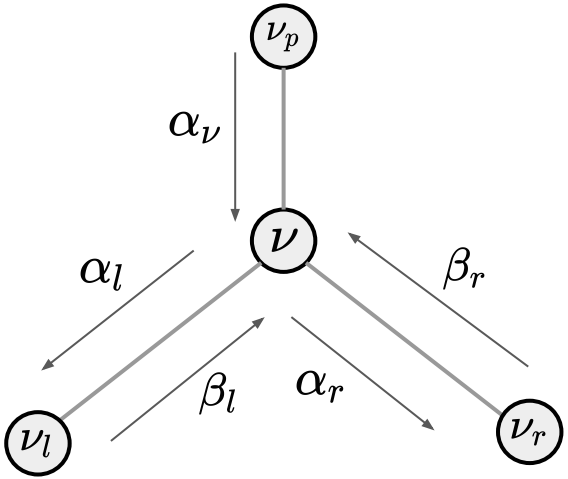
Successive Cancellation Decoding: Processing element



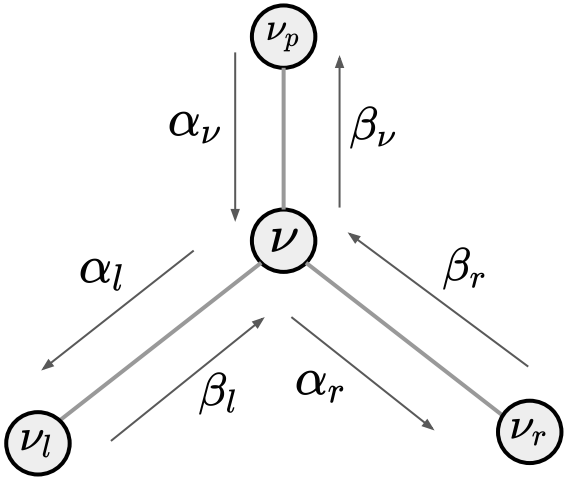
Successive Cancellation Decoding: Processing element



Successive Cancellation Decoding: Processing element



Successive Cancellation Decoding: Processing element

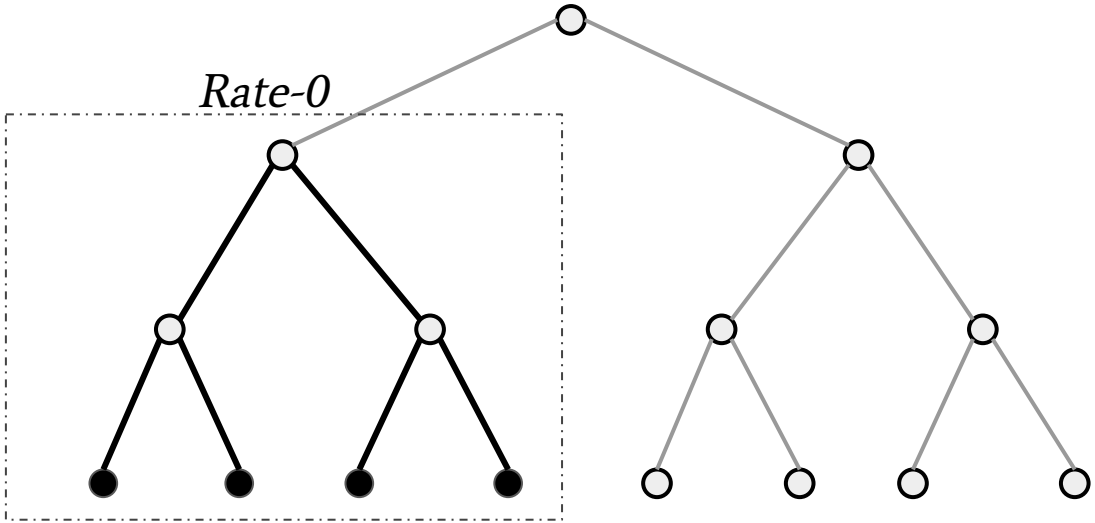


Simplified Successive Cancellation (SSC) Decoding

- Starting at the root node, traverse the binary tree in-order and at each node perform one of the following operations:
  - If a special node is encountered, stop and decode and move to parent node
    - This operation continues till you reach the first leaf node
  - If current node has a left child that was not visited, calculate α_l and move to left child
  - If current node has a right child that was not visited, calculate α_r and move to right child
  - If both the messages from child nodes are available, calculate β and move to parent node
- At root node, hard decision is made according to the binary quantizer function as:

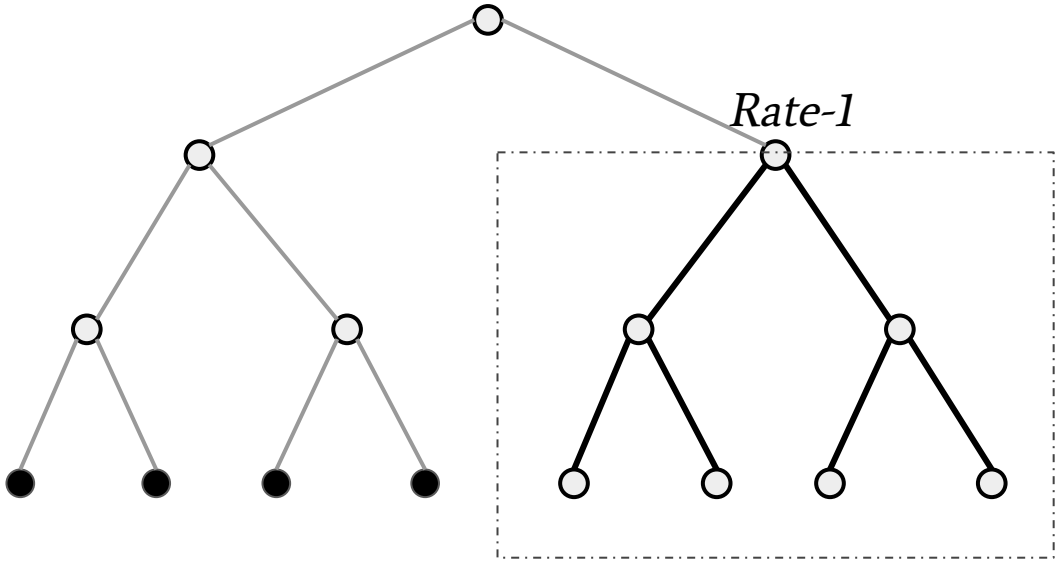
$$\beta_\nu = h(\alpha_\nu) = \begin{cases} 0, & \text{if } \alpha_\nu \geq 0 \\ 1, & \text{else} \end{cases}$$

SSCD for Rate-0 node



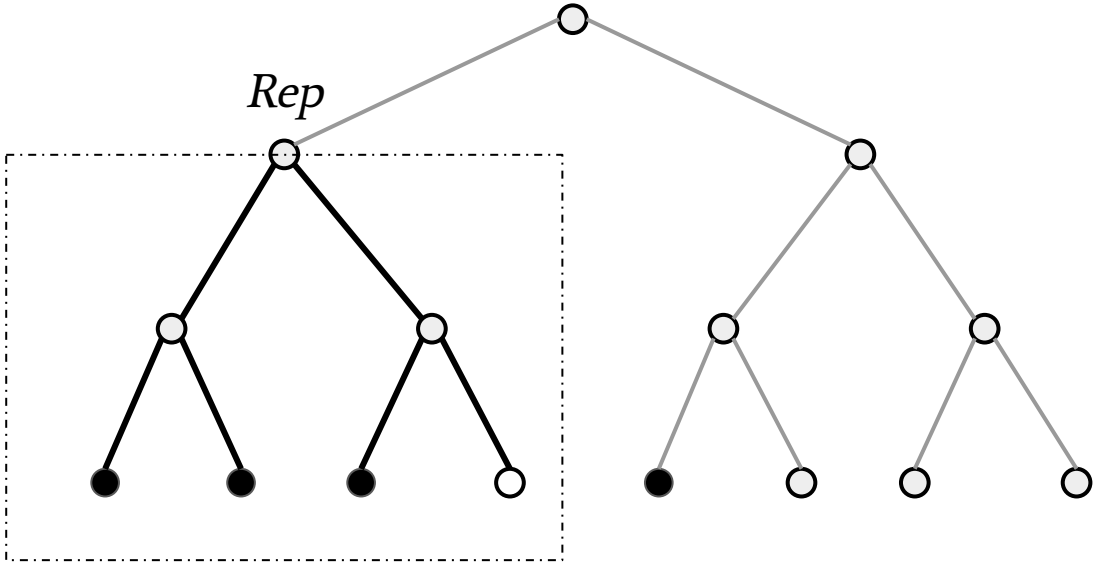
$$\beta_\nu = 0$$

SSCD for Rate-1 node



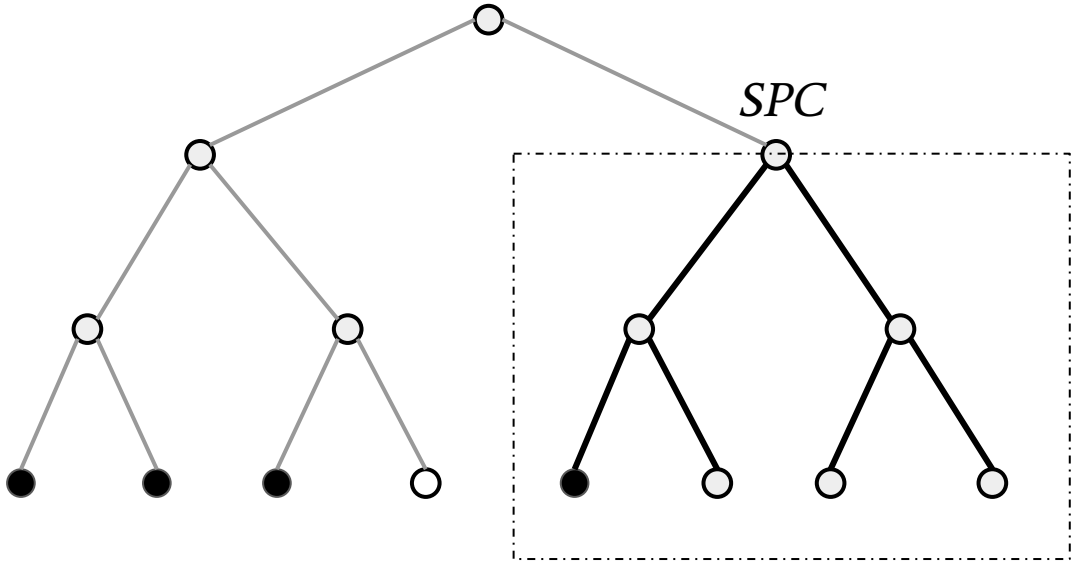
$$\beta_\nu = h(\alpha_\nu)$$

SSCD for Repetition node



$$\beta_\nu = h\left(\sum_{i=0}^{N_\nu} \alpha_\nu(i) \right)$$

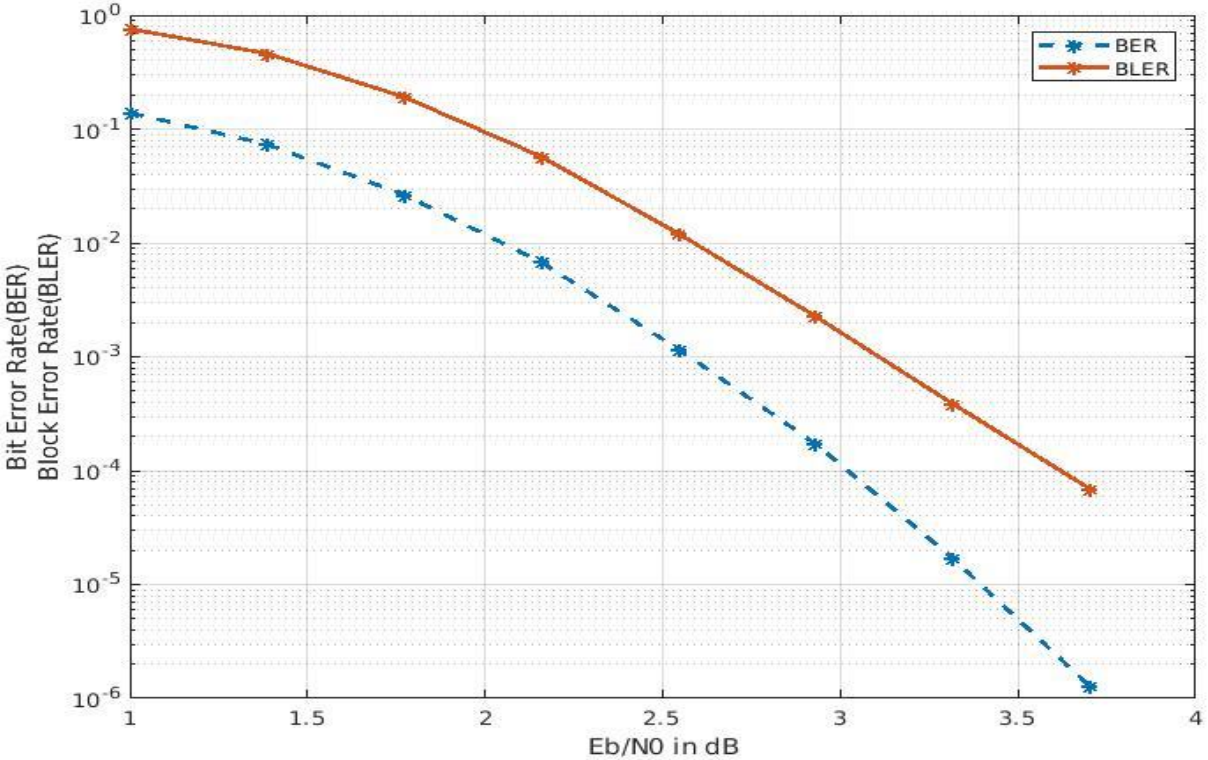
SSCD for Single Parity Check node



$$\beta_\nu = h(\alpha_\nu)$$

$$\beta_\nu(i_{par}) = \sum_{i=0; i \neq i_{par}}^{N_\nu} \beta_\nu(i)$$

BER/BLER for SC decoding (1024,512) polar code



Successive Cancellation List (SCL) Decoding

- The key idea is:
  - Each time a decision on β is needed, split the current decoding path into paths: β = 0 and 1
  - When the number of paths grows beyond a prescribed threshold *L,* discard the least probable paths and keep only the *L* best paths
  - At the end, select the most likely path

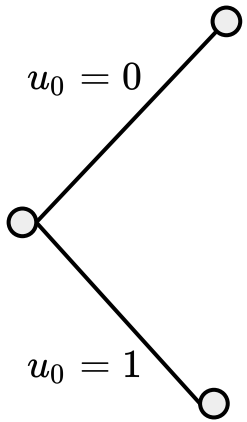
- Reliability for a path is indicated by assigning a *Path Metric* to each path and updates as:

$$PM_i = \begin{cases} PM_i, & \text{if } \beta = h(\alpha) \\ PM_i + |\alpha|, & \text{otherwise} \end{cases}$$

- The lesser the *Path Metric* the higher the reliability of a path

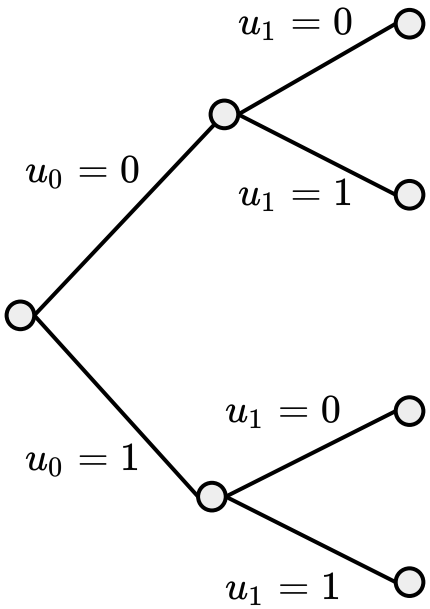
- CRC-SCL:
  - Concatenate the message vector with CRC bits
  - Pick the most likely path that satisfies CRC criteria

Successive Cancellation List Decoding: L = 4



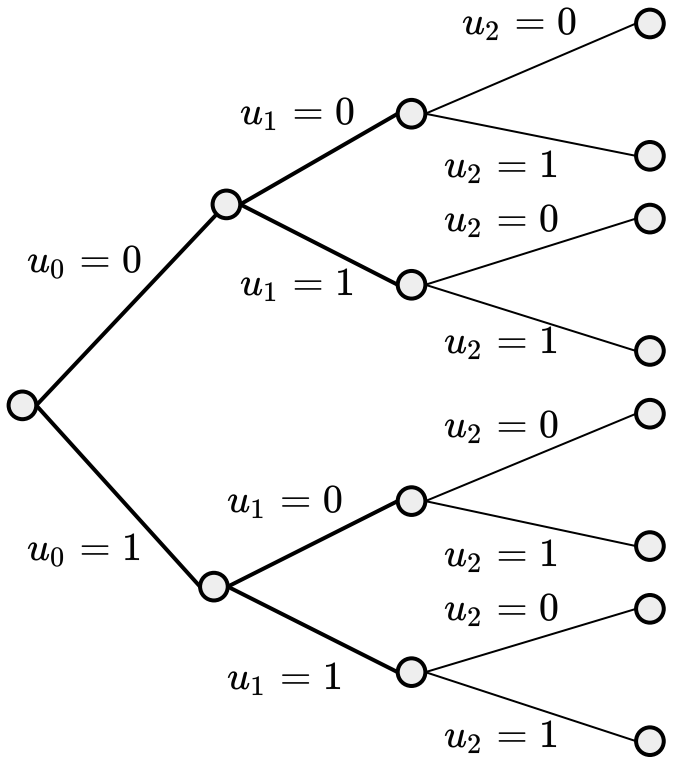
$$u_0 = \begin{cases} 0 \\ 1 \end{cases}$$

Successive Cancellation List Decoding: L = 4

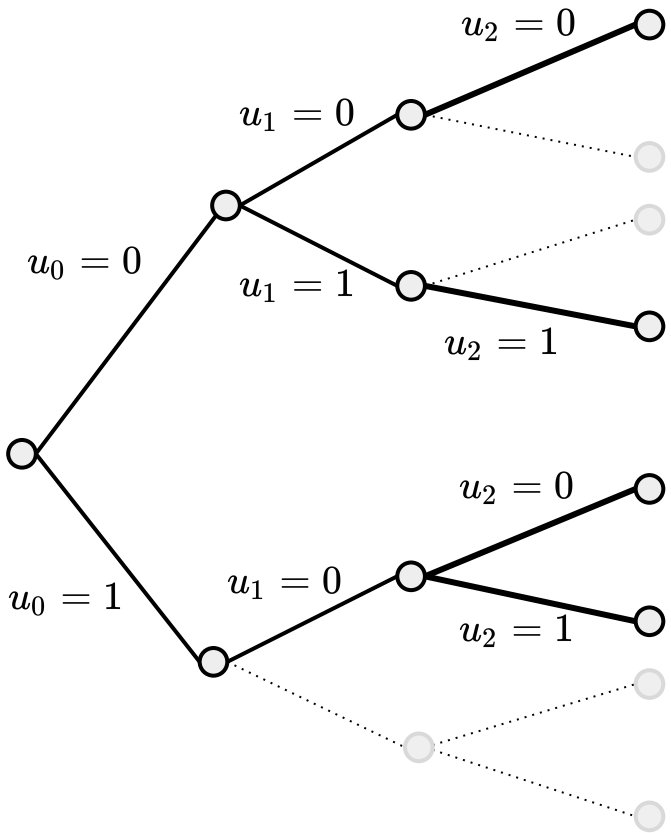


$$u_0 u_1 = \begin{cases} 00 \\ 01 \\ 10 \\ 11 \end{cases}$$

Successive Cancellation List Decoding: L = 4

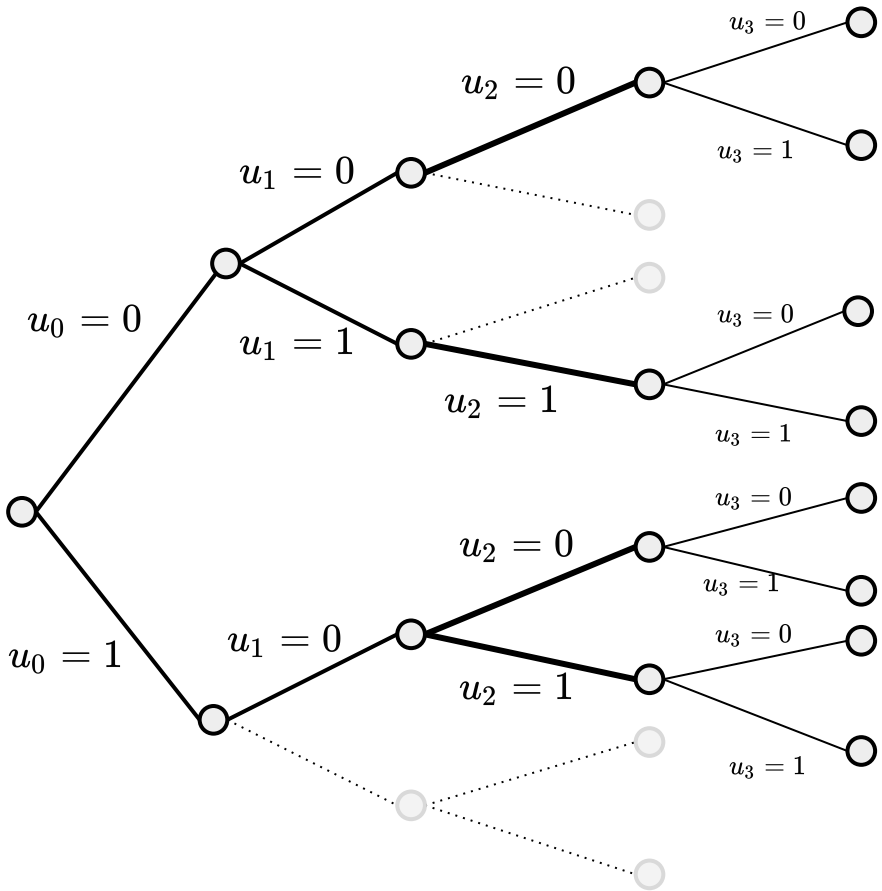


Successive Cancellation List Decoding: L = 4

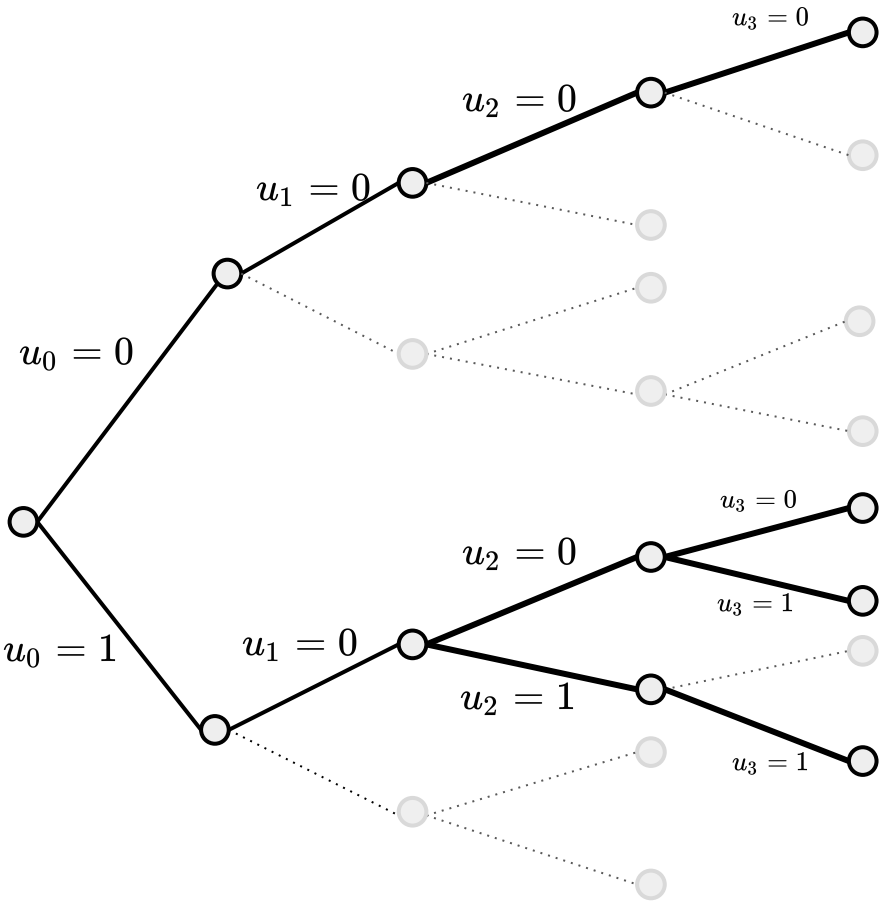


$$u_0 u_1 u_2 = \begin{cases} 000 \\ 011 \\ 100 \\ 101 \end{cases}$$

Successive Cancellation List Decoding: L = 4

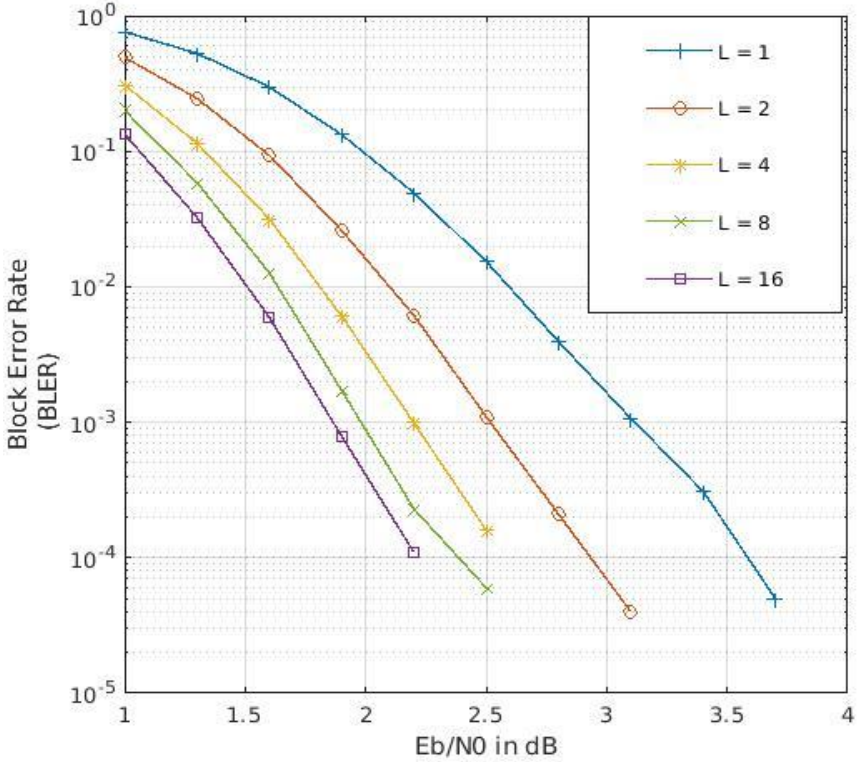


Successive Cancellation List Decoding: L = 4

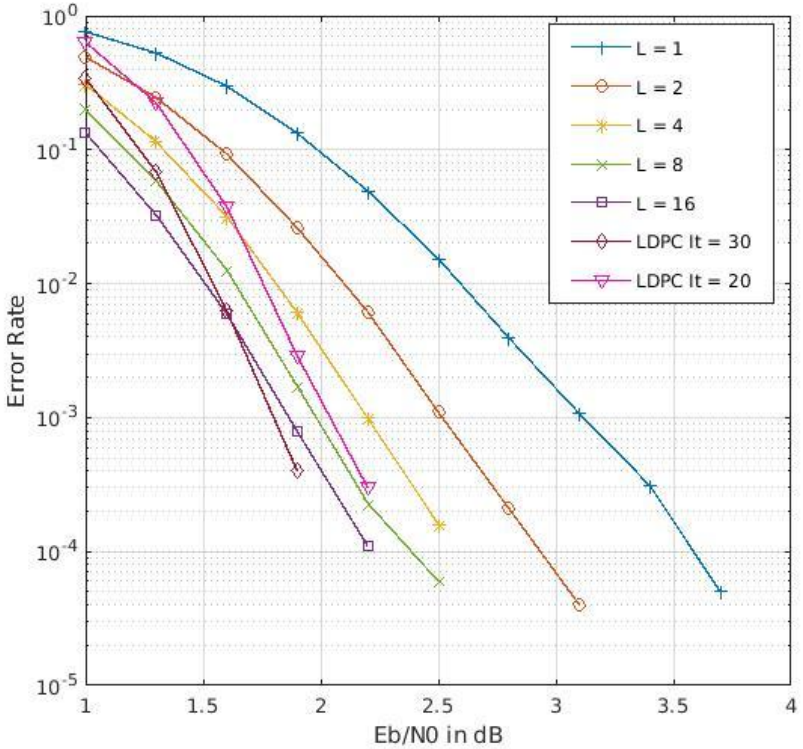


$$u_0 u_1 u_2 u_3 = \begin{cases} 0000 \\ 1000 \\ 1001 \\ 1011 \end{cases}$$

BLER for SCL decoding (1024,512,8) polar code



Comparison of (1944,972) LDPC code with (1024,512,8) polar code



Simplified SCL Decoding

- Decode special nodes directly without traversing the tree further
  - Custom PM update rules for different nodes
  - Reduced time steps per bit, possibility to decode multiple bits at once
- Provides significant improvement in throughput over SCL
- More scope for parallelisation as decoding of special nodes can be processed by special computing units

Fast-SSCL

- Limit the number of path splitting operations based on the list size
  - Rate-1 : *min(L-1,N)*
  - SPC: *min(L,N)*
- Based on the assumption that after decoding a certain number of bits, we have enough confidence to decode the rest of the bits separately
- At each special node, after the minimum number of path splitting operations are performed, remaining bits can be directly estimated using binary quantiser function

Encoder and Decoder information throughput results

- x86 processor - Intel i7-8700 @ 3.2GHz Information Throughput (IThp)

Encoder	**IThp**
Non-systematic	336 mbps
Systematic	24 mbps

Decoder	**IThp**
SCD	34 mbps
SSCD	93 mbps

Decoder	**L = 1**	**L = 2**	**L = 4**	**L = 8**	**L = 16**
SCL	18.1 mbps	10.3 mbps	5.6 mbps	2.5 mbps	1.3 mbps
SSCL	31 mbps	16.8 mbps	8.9 mbps	5.1 mbps	2.7 mbps

Decoder throughput results

- ARM processor - @ 1.2GHz

ENcoder	**Without NEON**	**With NEON**
Non-systematic	35 mbps	69 mbps
Systematic	2.5 mbps	5 mbps

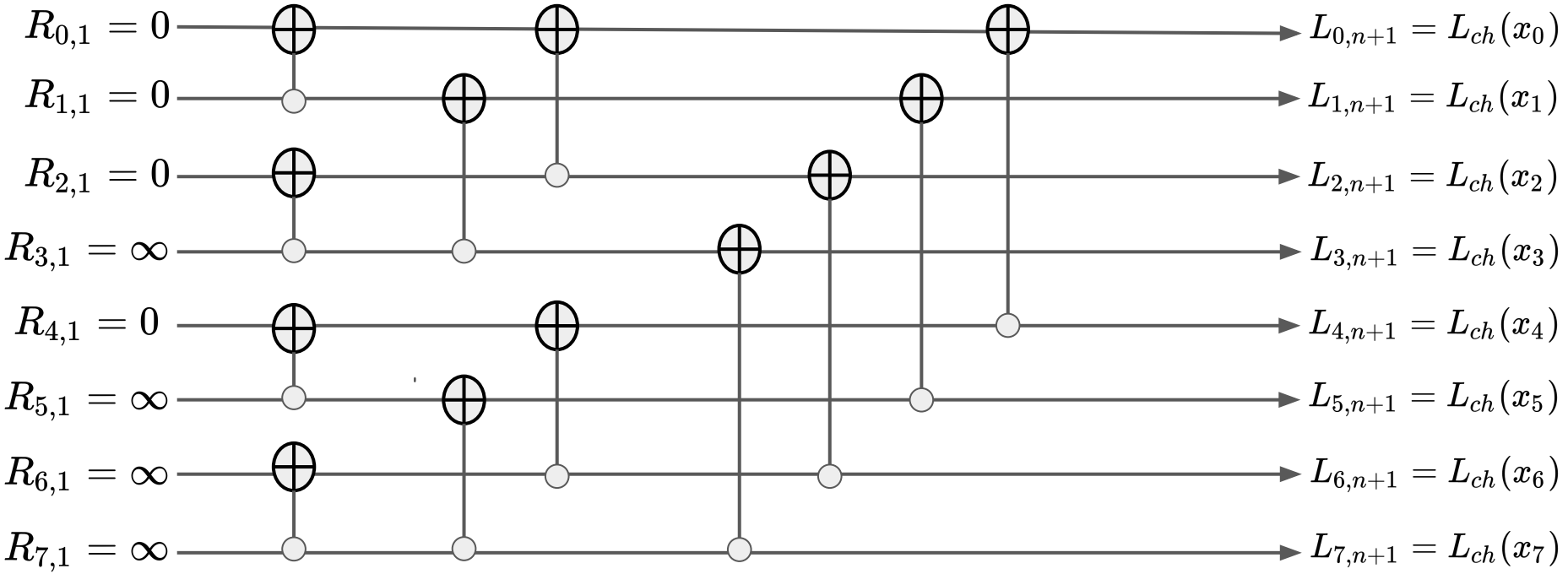
Decoder	**Without NEON**	**With NEON**
SCD	4.1	4.7
SSCD	9.9	13.9

Decoder	**L = 1**	**L = 2**	**L = 4**	**L = 8**	**L = 16**
SCL	1.7 mbps	1.1 mbps	0.6 mbps	0.3 mbps	0.14 mbps
SCL with NEON	2.1 mbps	1.4 mbps	0.8 mbps	0.4 mbps	0.21 mbps
SSCL	3.5 mbps	2.1 mbps	1.1 mbps	0.5 mbps	0.24 mbps
SSCL with NEON	4.2 mbps	2.7 mbps	1.4 mbps	0.7 mbps	0.36 mbps

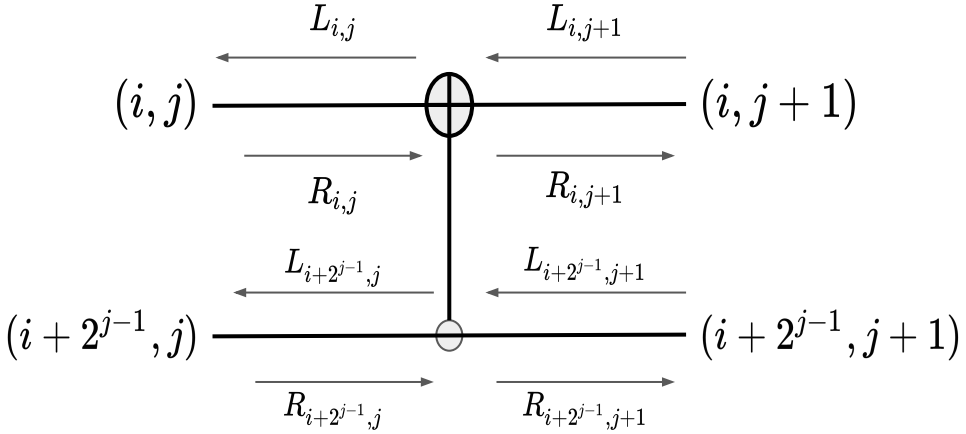
Belief Propagation decoding of Polar Codes

- Used traditionally for decoding LDPC Codes
- Much higher parallelisation possible than Successive Cancellation decoding
- Error correction performance comparable to SC decoding but much poorer compared to list decoding
- Reuse of optimized hardware architecture is the main advantage
- Two possible methods:
  - Iterating on encoder factor graph
  - Constructing tanner graph for parity check matrix

Arikan's BP decoder - encoder factor graph : (8,4) polar code



Arikan's BP decoder : Processing element



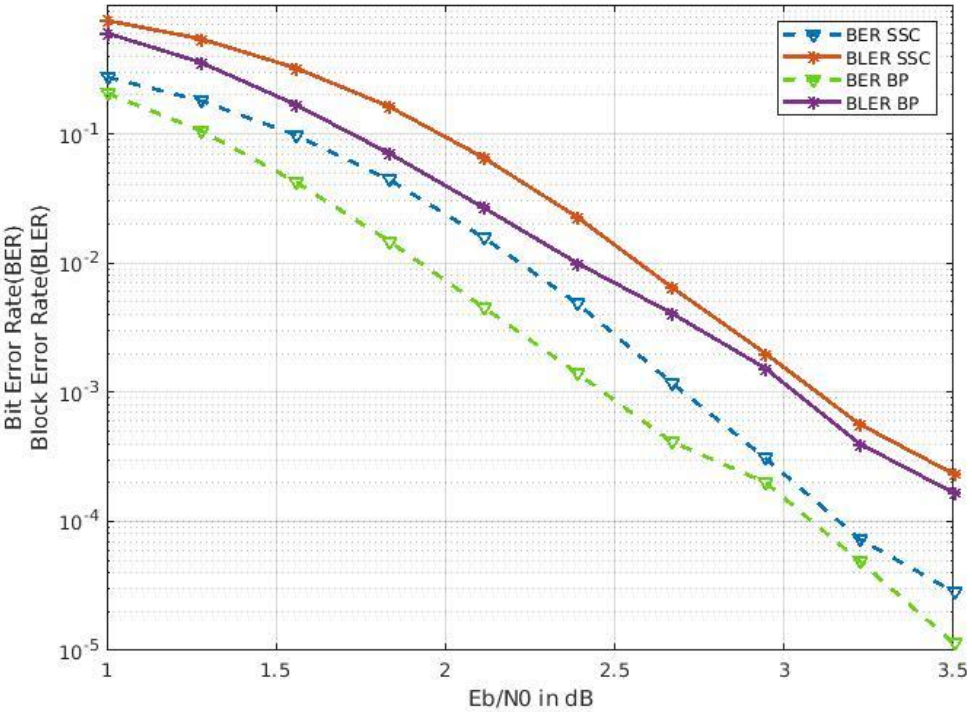
Update rules:

$$
\begin{aligned}
L_{i,j} &= f(L_{i,j+1}, (L_{i+2^{j-1},j+1} + R_{i+2^{j-1},j})) \\
L_{i+2^{j-1},j} &= f(R_{i,j}, L_{i,j+1}) + L_{i+2^{j-1},j+1} \\
R_{i,j+1} &= f(R_{i,j}, (L_{i+2^{j-1},j+1} + R_{i+2^{j-1},j})) \\
R_{i+2^{j-1},j+1} &= f(R_{i,j}, L_{i,j+1}) + R_{i+2^{j-1},j}
\end{aligned}
$$

Sparse graph based BP decoder

- Construct parity check *H* and the tanner graph from generator matrix *G*
- The effectiveness of BP decoder depends on the convergence achievable for the beliefs propagated
- To improve this, we prune the tanner graph to remove short cycles
- The new *H,* along with the received *LLR* is passed to a standard BP decoder for LDPC codes

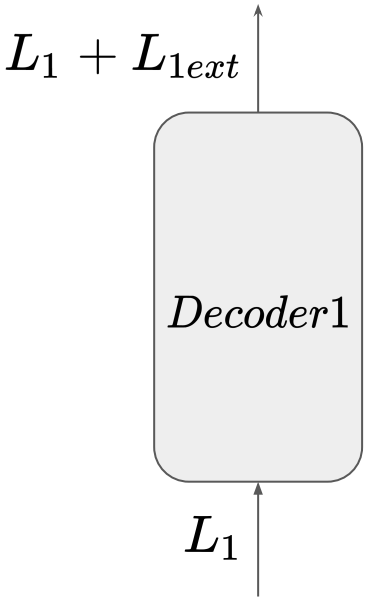
BER/BLER for BP decoding (1024,512) polar code



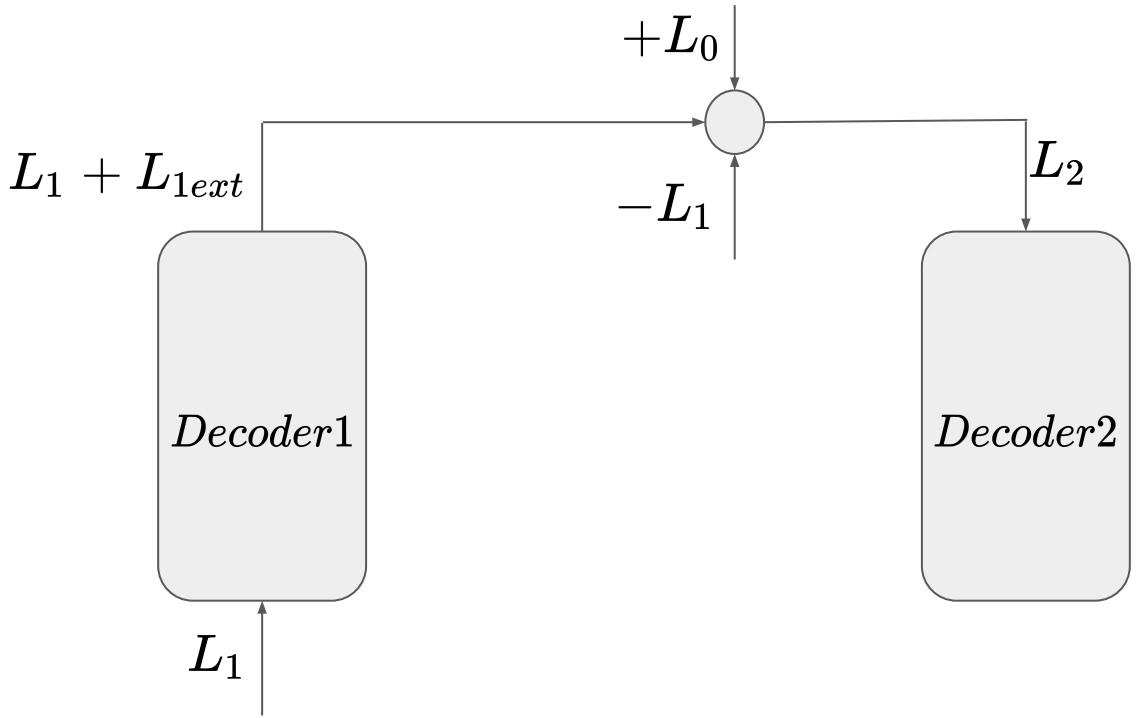
Concatenated BP decoder

- Add parity bits to the encoded vector to introduce more constraints
- Alternate between decoder for polar code and the decoder for the parity constraint passing extrinsic information to each other
- The first decoder corresponds to the polar code and the second decoder corresponds to the parity bits
- These parity bits can be used to improve the performance of data bits that are reliable but not the best
- Provides improvement in BER and BLER over the Sparse BP with a trade off in complexity and rate

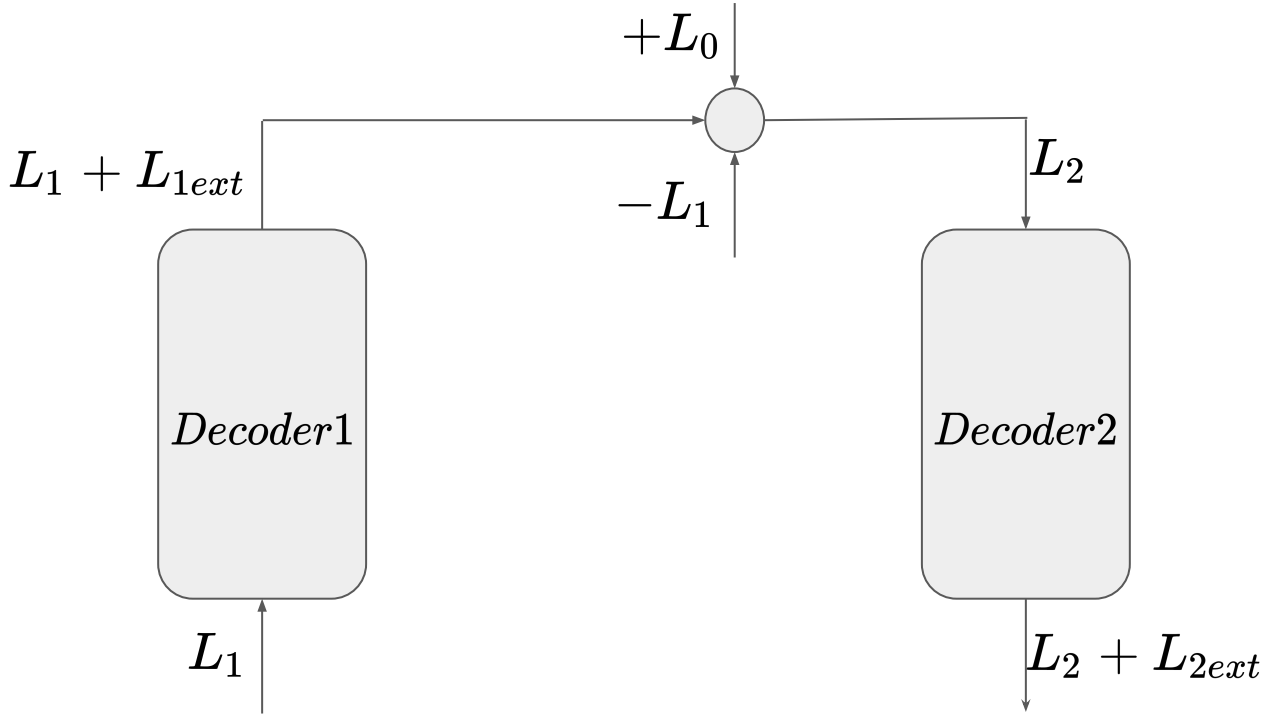
Concatenated BP decoder



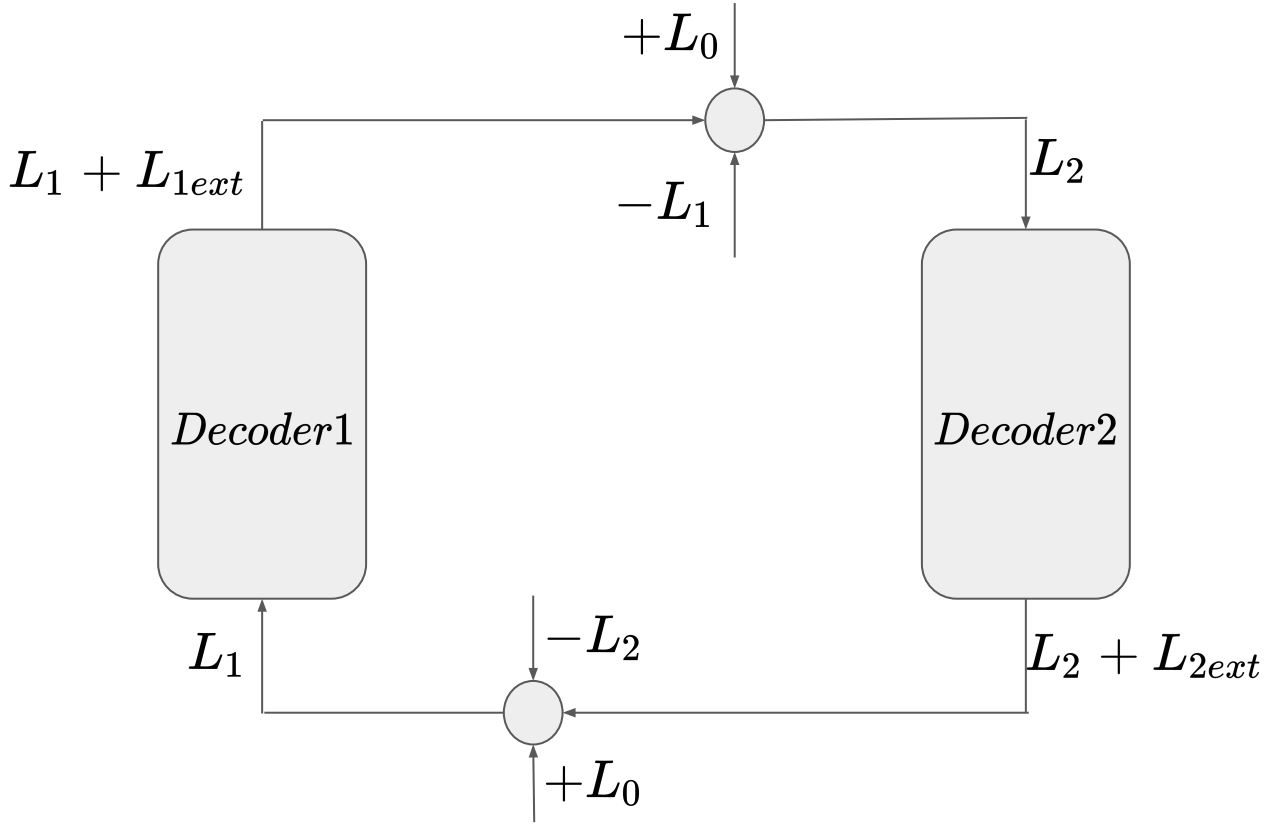
Concatenated BP decoder



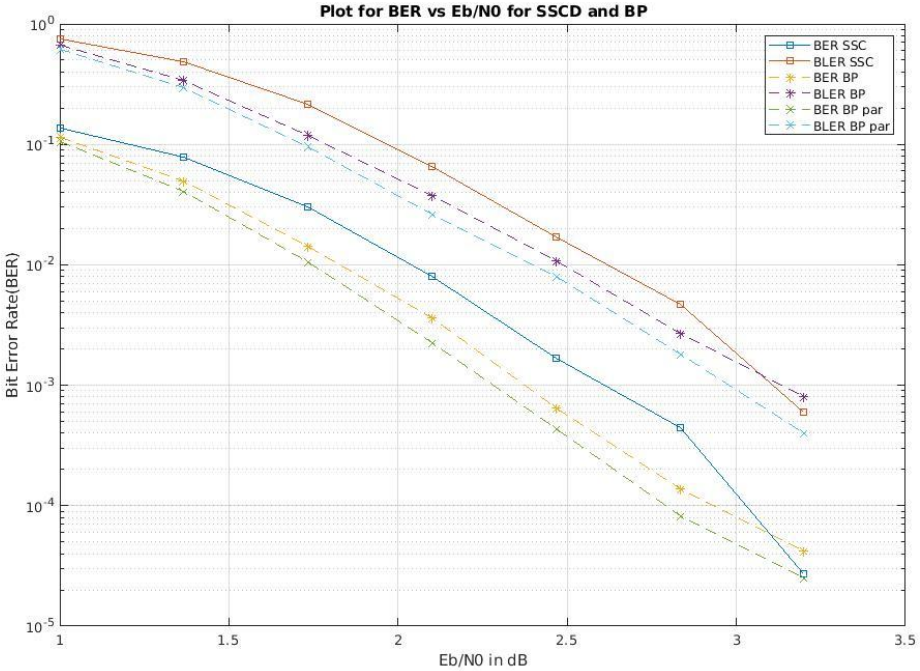
Concatenated BP decoder



Concatenated BP decoder



BER/BLER for concatenated BP decoding (1024,512) polar code



Future Work

- Fast-SSCL algorithm can be tuned to adaptively limit the maximum number of path splitting operations based on SNR
- Also Fast-SSCL algorithm provides a much larger scope for parallelisation than a SSCL algorithm because of the special nodes and custom hardware processing units can be built to leverage this
- Sparse graph based BP decoding has performance comparable to SC decoding and hence if the concept of list decoding can be used here, focusing on the least reliable bits, it can provide a significant improvement in performance
- Using concatenated codes, LDPC+Polar can provide a tradeoff between complexity and performance
    - A good thrust can be the design of the helper LDPC code that effectively focuses on the bits of Polar code most prone to error

THANK YOU!