Communications Hardware In The Post-Happy-Scaling Era

Alexios Balatsoukas-Stimming
Telecommunications Circuits Laboratory
École polytechnique fédérale de Lausanne
Switzerland

May 16, 2018

Electrical and Computer Engineering Department
Technical University of Crete
Acknowledgments

Funding:

Collaborators:
- W. J. Gross, S. A. Hashemi (McGill University)
- J. R. Cavallaro (Rice University)
- G. Matz, M. Meidlinger (TU Vienna)
- T. Podzorny, J. Uythoven (CERN)
- C.-H. Chen, F. Sheikh (Intel Labs)
The End of the “Happy Scaling” Era

- Increasing DSP algorithm complexity.
- Increasing need for flexibility.
- Increasing need for energy-efficiency.
- Severe variability & reliability issues.
- Vanishing energy & performance gains.
- Skyrocketing NRE: mask & design cost.

Algorithm/hardware co-design is more pertinent than ever. Maintaining progress will require cross-layer and interdisciplinary innovation.
The End of the “Happy Scaling” Era

Increasing DSP algorithm complexity.

Increasing need for flexibility.

Increasing need for energy-efficiency.

Severe variability & reliability issues.

Vanishing energy & performance gains.

Skyrocketing NRE: mask & design cost.

The Way Forward

1. Algorithm/hardware co-design is more pertinent than ever
2. Maintaining progress will require cross-layer and interdisciplinary innovation
Outline

1. **Classical algorithm/hardware co-design:**
   - Hardware implementation of successive cancellation list decoding of polar codes
   - Successive cancellation flip decoding of polar codes & its hardware implementation

2. **Approximate computing:**
   - Throughput-oriented construction of polar codes
   - Error-correction coding on faulty hardware

3. **Communications hardware meets information theory and machine learning:**
   - Terabit/s LDPC code decoders via quantized message passing
   - Neural networks for self-interference cancellation in full-duplex radios
Technology Innovations in 5G

- Wide Bandwidth and Carrier Aggregation
- Flexible and Scalable OFDMA Air-Interface
- Massive MIMO
- Small Cells and Advanced Cellular Concepts
- New Radio Frequencies (mmWave)
- Advanced Channel Codes (LDPC and polar)
Polar Codes

\[ A = \{3, 5, 6, 7\} \]

- **Construction:**
  - **Information indices:** \( A \subset \{0, 1, \ldots, N - 1\}, \ |A| = NR, \ N = 2^n. \)
Polar Codes

\[
\mathbf{u} = \begin{bmatrix}
  u_0 \\
  u_1 \\
  u_2 \\
  u_3 \\
  u_4 \\
  u_5 \\
  u_6 \\
  u_7 \\
\end{bmatrix}
\]

- **Construction:**
  - Information indices: \( \mathcal{A} \subset \{0, 1, \ldots, N - 1\} \), \( |\mathcal{A}| = NR \), \( N = 2^n \).

- **Encoding:**
  - \( \mathbf{u}_\mathcal{A} \triangleq [u_i, i \in \mathcal{A}]^T \leftarrow \text{data bits} \),
Polar Codes

\[ u = \begin{bmatrix} 0 \\ 0 \\ 0 \\ u_3 \\ 0 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} \]

- **Construction:**
  - Information indices: \( A \subset \{0, 1, \ldots, N - 1\} \), \( |A| = NR \), \( N = 2^n \).

- **Encoding:**
  - \( u_A \triangleq [u_i, i \in A]^T \leftarrow \) data bits,
  - \( u_{AC} \leftarrow \) known-to-receiver frozen bits (say all-zero).
Polar Codes

\[ u = \begin{bmatrix}
0 \\
0 \\
w_3 \\
0 \\
w_5 \\
w_6 \\
w_7
\end{bmatrix} \]

Encoder \[ x = Gu \]

- **Construction:**
  - Information indices: \( A \subset \{0, 1, \ldots, N - 1\}, |A| = NR, N = 2^n. \)

- **Encoding:**
  - \( u_A \triangleq [u_i, i \in A]^T \leftarrow \) data bits,
  - \( u_{AC} \leftarrow \) known-to-receiver frozen bits (say all-zero).
  - \( x \leftarrow Gu \) (using \( O(N \log N) \) binary additions).
Polar Codes

\[
\begin{bmatrix}
0 \\
0 \\
0 \\
u_3 \\
0 \\
u_5 \\
u_6 \\
u_7
\end{bmatrix}
\]

Encoder: \( x = Gu \)

\[
W^N(y|x) = \prod_{i=1}^{N} W(y_i|x_i)
\]

• Construction:
  - Information indices: \( A \subset \{0, 1, \ldots, N-1\} \), \( |A| = NR \), \( N = 2^n \).

• Encoding:
  - \( u_A \triangleq [u_i, i \in A]^T \leftarrow \) data bits,
  - \( u_{AC} \leftarrow \) known-to-receiver frozen bits (say all-zero).
  - \( x \leftarrow Gu \) (using \( O(N \log N) \) binary additions).
Polar Codes

\[ u = \begin{bmatrix} 0 \\ 0 \\ 0 \\ u_3 \\ 0 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} \]

Encoder: \( x = G u \)

Memoryless Channel: \( W_N(y|x) = \prod_{i=1}^{N} W(y_i|x_i) \)

Decoder: \( \hat{u} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \hat{u}_3 = ? \\ 0 \\ \hat{u}_5 = ? \\ \hat{u}_6 = ? \\ \hat{u}_7 = ? \end{bmatrix} \)

- Construction:
  - Information indices: \( A \subset \{0, 1, \ldots, N - 1\} \), \( |A| = NR, \ N = 2^n \).

- Encoding:
  - \( u_A \triangleq [u_i, i \in A]^T \leftarrow \) data bits,
  - \( u_{AC} \leftarrow \) known-to-receiver frozen bits (say all-zero).
  - \( x \leftarrow G u \) (using \( O(N \log N) \) binary additions).

- Decoding: Estimate the information bits \( \hat{u}_A \).
**Polar Codes**

\[
\begin{bmatrix}
0 \\
0 \\
0 \\
u_3 \\
u_5 \\
u_6 \\
u_7
\end{bmatrix}
\xrightarrow{\text{Encoder}}
\begin{bmatrix}
\begin{bmatrix}
x \in \{0, 1\}^N
\end{bmatrix}
\end{bmatrix}
\xrightarrow{\text{Memoryless Channel}}
\begin{bmatrix}
y \in \mathcal{Y}^N
\end{bmatrix}
\xrightarrow{\text{Decoder}}
\begin{bmatrix}
\hat{u}_3 = ? \\
\hat{u}_5 = ? \\
\hat{u}_6 = ? \\
\hat{u}_7 = ?
\end{bmatrix}
\]

\[
\begin{align*}
\text{arg max } & \Pr[Y = y, U_i^j = \hat{u}_i | U_i = \hat{u}_i] \\
\text{Construction:} & \quad \text{Information indices: } \mathcal{A} \subset \{0, 1, \ldots, N - 1\}, \ |\mathcal{A}| = NR, \ N = 2^n.
\end{align*}
\]

**Encoding:**
- \( u_\mathcal{A} \triangleq [u_i, i \in \mathcal{A}]^T \leftarrow \text{data bits}, \\
- \( u_\mathcal{A}^C \leftarrow \text{known-to-receiver frozen bits (say all-zero)}, \\
- \( x \leftarrow G u \) (using \( O(N \log N) \) binary additions).

**Decoding:** Estimate the information bits \( \hat{u}_\mathcal{A} \).
- **Successive Cancellation (SC) Decoding:** At each level \( i \in \mathcal{A} \), choose the best possible value of \( u_i \) given the past estimations and frozen bits.
Successive Cancellation Decoding

- Successive traversal of a data dependency graph
- $N \log N$ nodes, each visited exactly once $\rightarrow O(N \log N)$ time complexity!
- Re-use of memory positions $\rightarrow O(N)$ memory complexity!

\[
\begin{align*}
\hat{u} & \quad S_0 & \quad S_1 & \quad S_2 & \quad y \\
\hat{u}_0 = 0 & \quad + & \quad u_0 & \quad + & \quad y_0 \\
\hat{u}_1 = 0 & \quad + & \quad u_0 & \quad + & \quad y_1 \\
\hat{u}_2 = 0 & \quad + & \quad u_0 \oplus u_1 & \quad + & \quad y_2 \\
\hat{u}_3 = \hat{a}_0 & \quad + & \quad u_2 & \quad + & \quad y_3 \\
\hat{u}_4 = 0 & \quad + & \quad u_2 \oplus u_3 & \quad + & \quad y_4 \\
\hat{u}_5 = \hat{a}_1 & \quad + & \quad u_4 & \quad + & \quad y_5 \\
\hat{u}_6 = \hat{a}_2 & \quad + & \quad u_4 \oplus u_5 & \quad + & \quad y_6 \\
\hat{u}_7 = \hat{a}_3 & \quad + & \quad u_6 & \quad + & \quad y_7
\end{align*}
\]
**Successive Cancellation Decoding**

- Successive traversal of a data dependency graph
- $N \log N$ nodes, each visited exactly once $\rightarrow O(N \log N)$ time complexity!
- Re-use of memory positions $\rightarrow O(N)$ memory complexity!

\[\begin{array}{c}
\hat{u} \\
\hat{u}_0 = 0 \\
\hat{u}_1 = 0 \\
\hat{u}_2 = 0 \\
\hat{u}_3 = \hat{a}_0 \\
\hat{u}_4 = 0 \\
\hat{u}_5 = \hat{a}_1 \\
\hat{u}_6 = \hat{a}_2 \\
\hat{u}_7 = \hat{a}_3 \\
\end{array}\]

\[\begin{array}{c}
S_0 \\
\frac{\hat{a}_0}{\hat{u}_0} \\
\frac{\hat{u}_1}{u_0} \\
\frac{\hat{u}_2}{\hat{a}_0 \oplus u_1} \\
\frac{\hat{u}_3}{\hat{a}_0 \oplus \hat{a}_1} \\
\frac{\hat{u}_4}{\hat{a}_0 \oplus \hat{a}_1 \oplus u_2} \\
\frac{\hat{u}_5}{\hat{a}_0 \oplus \hat{a}_1 \oplus u_2 \oplus u_3} \\
\frac{\hat{u}_6}{\hat{a}_0 \oplus \hat{a}_1 \oplus u_2 \oplus u_3} \\
\frac{\hat{u}_7}{\hat{a}_0 \oplus \hat{a}_1 \oplus u_2 \oplus u_3} \\
\end{array}\]

\[\begin{array}{c}
y \\
y_0 \\
y_1 \\
y_2 \\
y_3 \\
y_4 \\
y_5 \\
y_6 \\
y_7 \\
\end{array}\]

- Two simple soft information update operations:

\[f(\alpha_i, \alpha_j) \approx \text{sgn}(\alpha_i)\text{sgn}(\alpha_j)\min(|\alpha_i|, |\alpha_j|)\]

\[g(\alpha_i, \alpha_j, \hat{a}_k) = \alpha_j + (-1)^{\hat{a}_k} \alpha_i\]
Hardware Implementation of SC Decoding

- **Decoder Core**: contains $P$ processing elements that implement update rules
- **Memories**: store soft information, partial sums, and decoded codeword
- **Controller**: organizes memory reads and writes, and update rule selection
Hardware Implementation of SC Decoding

- **Decoder Core**: contains $P$ processing elements that implement update rules
- **Memories**: store soft information, partial sums, and decoded codeword
- **Controller**: organizes memory reads and writes, and update rule selection

- **Simple** and **flexible** architecture
- **Compact** and **energy-efficient**
Hardware Implementation of SC Decoding

- **Decoder Core**: contains $P$ processing elements that implement update rules
- **Memories**: store soft information, partial sums, and decoded codeword
- **Controller**: organizes memory reads and writes, and update rule selection

**Two main challenges with SC decoding:**

1. Low throughput due to sequential nature
2. Mediocre error-correcting performance due to error propagation
Hardware Implementation of SC Decoding

- **Decoder Core**: contains $P$ processing elements that implement update rules
- **Memories**: store soft information, partial sums, and decoded codeword
- **Controller**: organizes memory reads and writes, and update rule selection

Two main challenges with SC decoding:

1. Low throughput due to sequential nature
2. Mediocre error-correcting performance due to error propagation

- Simple and flexible architecture
- Compact and energy-efficient
Successive Cancellation List Decoding

- **SC Decoding**: past errors can never be corrected
- **SCL Decoding**: up to $L$ simultaneous paths on the decoding tree
  - **Time complexity**: $O(LN \log N)$, **Memory complexity**: $O(LN)$

---

## Successive Cancellation List Decoding

- **SC Decoding**: past errors can never be corrected
- **SCL Decoding**: up to \( L \) simultaneous paths on the decoding tree

  - **Time complexity**: \( O(LN \log N) \)
  - **Memory complexity**: \( O(LN) \)

\[
N = 8, \ A = \{3, 5, 6, 7\}
\]

---

Successive Cancellation List Decoding

- **SC Decoding**: past errors can never be corrected
- **SCL Decoding**: up to $L$ simultaneous paths on the decoding tree
  - **Time complexity**: $O(LN \log N)$, **Memory complexity**: $O(LN)$

- $N = 8$, $\mathcal{A} = \{3, 5, 6, 7\}$
- **SC Decoder**

---

Successive Cancellation List Decoding

- **SC Decoding**: past errors can never be corrected
- **SCL Decoding**: up to $L$ simultaneous paths on the decoding tree
  - Time complexity: $O(LN \log N)$, Memory complexity: $O(LN)$

- $N = 8$, $A = \{3, 5, 6, 7\}$

---

**Successive Cancellation List Decoding**

- **SC Decoding**: past errors can never be corrected
- **SCL Decoding**: up to $L$ simultaneous paths on the decoding tree
  - **Time complexity**: $O(LN \log N)$, **Memory complexity**: $O(LN)$

- $N = 8$, $\mathcal{A} = \{3, 5, 6, 7\}$

---

Successive Cancellation List Decoding

- **SC Decoding**: past errors can never be corrected
- **SCL Decoding**: up to $L$ simultaneous paths on the decoding tree
  - *Time complexity*: $O(LN \log N)$, *Memory complexity*: $O(LN)$

- $N = 8$, $\mathcal{A} = \{3, 5, 6, 7\}$

---

Successive Cancellation List Decoding

- **SC Decoding**: past errors can never be corrected
- **SCL Decoding**: up to $L$ simultaneous paths on the decoding tree
  - **Time complexity**: $O(LN \log N)$, **Memory complexity**: $O(LN)$

- $N = 8$, $A = \{3, 5, 6, 7\}$

---

Successive Cancellation List Decoding

- **SC Decoding:** past errors can never be corrected
- **SCL Decoding:** up to $L$ simultaneous paths on the decoding tree
  - Time complexity: $O(LN \log N)$, Memory complexity: $O(LN)$

- $N = 8$, $\mathcal{A} = \{3, 5, 6, 7\}$

---

Successive Cancellation List Decoding

- **SC Decoding**: past errors can never be corrected
- **SCL Decoding**: up to $L$ simultaneous paths on the decoding tree
  - **Time complexity**: $O(LN \log N)$, **Memory complexity**: $O(LN)$

$N = 8$, $\mathcal{A} = \{3, 5, 6, 7\}$

SC Decoder
- **SC List Decoder**, $L = 2$

---

**Successive Cancellation List Decoding**

- **SC Decoding**: past errors can never be corrected
- **SCL Decoding**: up to $L$ simultaneous paths on the decoding tree
  - Time complexity: $O(LN \log N)$, Memory complexity: $O(LN)$

- $N = 8$, $A = \{3, 5, 6, 7\}$
- **SC Decoder**
- **SC List Decoder**, $L = 2$

---

Successive Cancellation List Decoding

- **SC Decoding**: past errors can never be corrected
- **SCL Decoding**: up to $L$ simultaneous paths on the decoding tree
  - **Time complexity**: $O(LN \log N)$, **Memory complexity**: $O(LN)$

- $N = 8$, $A = \{3, 5, 6, 7\}$
- **SC Decoder**
- **SC List Decoder**, $L = 2$

---

Successive Cancellation List Decoding

- **SC Decoding**: past errors can never be corrected
- **SCL Decoding**: up to $L$ simultaneous paths on the decoding tree
  - Time complexity: $O(LN \log N)$, Memory complexity: $O(LN)$

- $N = 8$, $A = \{3, 5, 6, 7\}$

- SC Decoder
- SC List Decoder, $L = 2$

---

Successive Cancellation List Decoding

- **SC Decoding**: past errors can never be corrected
- **SCL Decoding**: up to $L$ simultaneous paths on the decoding tree
  - **Time complexity**: $O(LN \log N)$, **Memory complexity**: $O(LN)$

- $N = 8$, $A = \{3, 5, 6, 7\}$
- **SC Decoder**
- **SC List Decoder**, $L = 2$

---

Successive Cancellation List Decoding

- **SC Decoding**: past errors can never be corrected
- **SCL Decoding**: up to $L$ simultaneous paths on the decoding tree
  - **Time complexity**: $O(LN \log N)$, **Memory complexity**: $O(LN)$

- $N = 8$, $A = \{3, 5, 6, 7\}$
- **SC Decoder**
- **SC List Decoder**, $L = 2$

---

Successive Cancellation List Decoding

- **SC Decoding**: past errors can never be corrected
- **SCL Decoding**: up to \( L \) simultaneous paths on the decoding tree
  - Time complexity: \( O(LN \log N) \), Memory complexity: \( O(LN) \)

- \( N = 8, A = \{3, 5, 6, 7\} \)

- **SC Decoder**
- **SC List Decoder, \( L = 2 \)**

---

Hardware Implementation of SCL Decoding

What changes w.r.t. SC decoding?

- Perform computations for $L$ paths simultaneously
- Compute and sort path metrics to keep $L$ best paths at each step
Hardware Implementation of SCL Decoding

What changes w.r.t. SC decoding?

- Perform computations for $L$ paths simultaneously (highly parallelizable!)
- Compute and sort path metrics to keep $L$ best paths at each step

What we need:

1. $L$ decoder cores
2. $L$ SC memories
3. A path metric sorter
Hardware Implementation of SCL Decoding

What changes w.r.t. SC decoding?

- Perform computations for \( L \) paths simultaneously
- Compute and sort path metrics to keep \( L \) best paths at each step

What we need:

1. \( L \) decoder cores
2. \( L \) SC memories
3. A path metric sorter

We proved an arithmetic re-formulation of SCL decoding that makes the hardware implementation up to 67% more hardware-efficient!

Optimized Metric Sorting for SCL Decoding

- Metric sorter lies on the **critical path** of SCL decoders

Radix-2\(L\) Sorter

Bitonic Sorter

Bubble Sorter

Significant improvement in the area and operating frequency of the decoder!

---

Optimized Metric Sorting for SCL Decoding

- Metric sorter lies on the **critical path** of SCL decoders
- Exploit reformulated metric properties to simplify the sorter:
  1. When forking, the $L$ new path metrics are augmented versions of the old $L$ ones
  2. Just need the $L$ best among $2L$, no need for the $L$ best to be sorted

Radix-$2L$ Sorter

Bitonic Sorter

Bubble Sorter

Significant improvement in the area and operating frequency of the decoder!

Optimized Metric Sorting for SCL Decoding

- Metric sorter lies on the **critical path** of SCL decoders
- Exploit reformulated metric properties to simplify the sorter:
  1. When forking, the $L$ new path metrics are augmented versions of the old $L$ ones
  2. Just need the $L$ best among $2L$, no need for the $L$ best to be sorted

---

**Radix-$2L$ Sorter**

**Bitonic Sorter**

**Bubble Sorter**

---

Significant improvement in the **area and operating frequency** of the decoder!

---

Successive Cancellation Flip Decoding

SCL Decoding

Most of the computations and memory are wasted most of the time!
**Successive Cancellation Flip Decoding**

### SCL Decoding

Most of the computations and memory are **wasted most of the time**!

- **Observation**: Under SC, most faulty frames are the result of **one wrong decision**

![Graph showing relative frequency of number of errors for different Eb/N0 values](image-url)
Successive Cancellation Flip Decoding

SCL Decoding
Most of the computations and memory are wasted most of the time!

• Observation: Under SC, most faulty frames are the result of one wrong decision

• Successive Cancellation Flip (SCF) decoding:
  1. Perform SC decoding and track \( T \) most unreliable decisions
  2. Use a CRC to identify erroneous decoding
  3. Re-run SC up to \( T \) times, each time flipping the most unreliable decision

---

Successive Cancellation Flip Decoding

SCL Decoding

Most of the computations and memory are **wasted most of the time**!

- **Observation**: Under SC, most faulty frames are the result of **one wrong decision**

![Graph showing relative frequency vs. number of errors for different Eb/N0 values](image)

- **Successive Cancellation Flip (SCF) decoding**:
  1. Perform SC decoding and track $T$ most unreliable decisions
  2. Use a CRC to identify erroneous decoding
  3. Re-run SC up to $T$ times, each time flipping the most unreliable decision

Error-correcting performance in-between SC and SCL, but:

- **Memory complexity** of SC decoding
- **(Average) time complexity** of SC decoding (at high SNR)

---

Hardware Implementation of SCF Decoding

- **Simple:** Add an insertion sorter and a CRC unit to an SC decoder

Negligible area overhead!  
No impact on latency!
Hardware Implementation of SCF Decoding

• **Simple:** Add an insertion sorter and a CRC unit to an SC decoder

Negligible area overhead!
No impact on latency!

Our proposed SCF decoding algorithm is being considered by the 3GPP as an ultra-low power option for massive machine type communications for IoT in 5G

---

Bringing It All Together

- **PolarBear**: Manufactured ASIC in ST 28 nm FD-SOI
  - SC, SCF, and SCL decoding on the same chip
  - Run-time algorithm selection for energy-proportional operation

VLSI Circuits are Becoming Unreliable

- Devices suffer from **defects** due to **parameter variations** and "soft errors"
- These issues **compromise reliable operation** and **prevent effective power-reduction techniques** (e.g., voltage scaling)
VLSI Circuits are Becoming Unreliable

- Devices suffer from **defects** due to **parameter variations** and “soft errors”
- These issues **compromise reliable operation** and **prevent effective power-reduction techniques** (e.g., voltage scaling)
- Memories are **particularly sensitive to process variations** and **dominate area and power consumption** of modern systems-on-chip
VLSI Circuits are Becoming Unreliable

- Devices suffer from defects due to parameter variations and “soft errors”
- These issues compromise reliable operation and prevent effective power-reduction techniques (e.g., voltage scaling)
- Memories are particularly sensitive to process variations and dominate area and power consumption of modern systems-on-chip

What to do?

- Hardware protection to avoid errors is costly in terms of area and power
- Discarding faulty chips can decrease the yield dramatically
Approximate Computing

• **Fortunately**, many applications deal with data that is already **stochastic** and/or degrade **gracefully** when data is corrupted
Approximate Computing

- **Fortunately**, many applications deal with data that is already **stochastic** and/or degrade **gracefully** when data is corrupted.

- **Inherent application resilience** can also be exploited for algorithmic simplifications.
Approximate Computing

- **Fortunately**, many applications deal with data that is already **stochastic** and/or degrade **gracefully** when data is corrupted.
- **Inherent application resilience** can also be exploited for algorithmic simplifications.

**Example: error-correcting codes**

- Throughput-oriented construction of polar codes
- Faulty successive cancellation decoding of polar codes
- Faulty min-sum decoding of LDPC codes
- Faulty windowed min-sum decoding of spatially-coupled LDPC codes

2. A. Balatsoukas-Stimming and A. Burg, "Faulty successive cancellation decoding of polar codes for the binary erasure channel," IEEE Transactions on Communications, Dec. 2017
Approximate Computing

- **Fortunately**, many applications deal with data that is already **stochastic** and/or degrade **gracefully** when data is corrupted.
- **Inherent application resilience** can also be exploited for algorithmic simplifications.

**Example: error-correcting codes**

- Throughput-oriented construction of polar codes
- Faulty successive cancellation decoding of polar codes
- Faulty min-sum decoding of LDPC codes
- Faulty windowed min-sum decoding of spatially-coupled LDPC codes

---

2. A. Balatsoukas-Stimming and A. Burg, "Faulty successive cancellation decoding of polar codes for the binary erasure channel," IEEE Transactions on Communications, Dec. 2017
Throughput-Oriented Construction of Polar Codes

- Some SC decoding computations can be skipped for frozen bit groups
- Throughput of SC decoding depends on distribution of frozen bits
Throughput-Oriented Construction of Polar Codes

- Some SC decoding computations can be skipped for frozen bit groups
- Throughput of SC decoding depends on distribution of frozen bits
- Idea: maximize a weighted sum of the throughput and a performance metric
  - Integer linear program $\rightarrow$ greedy algorithm
Throughput-Oriented Construction of Polar Codes

- Some SC decoding computations can be skipped for frozen bit groups
- Throughput of SC decoding depends on the distribution of frozen bits
- Idea: maximize a weighted sum of the throughput and a performance metric
  - Integer linear program → greedy algorithm

Numerous complexity-performance trade-offs

Minimal performance degradation
Polar Codes over the Binary Erasure Channel

• Binary Erasure Channel (BEC):
  - **Input**: 0 or 1
  - **Output**: equal to the input with probability \(1 - p\), equal to ? with probability \(p\)
Polar Codes over the Binary Erasure Channel

• Binary Erasure Channel (BEC):
  - **Input**: 0 or 1
  - **Output**: equal to the input with probability $1 - p$, equal to ? with probability $p$

• Decoding over the BEC:
  - **Update rules**:
    - $f(a_i, a_j)$: output is ? if at least one input is ?
    - $g(a_i, a_j)$: output is ? if both inputs are ?
Polar Codes over the Binary Erasure Channel

• Binary Erasure Channel (BEC):
  ■ **Input:** 0 or 1
  ■ **Output:** equal to the input with probability $1 - p$, equal to $\oplus$ with probability $p$

• Decoding over the BEC:
  ■ **Update rules:**
    \[ f(a_i, a_j): \text{output is } \oplus \text{ if at least one input is } \oplus \]
    \[ g(a_i, a_j): \text{output is } \oplus \text{ if both inputs are } \oplus \]

• What is the erasure probability for each bit?
Polar Codes over the Binary Erasure Channel

- **Binary Erasure Channel (BEC):**
  - **Input:** 0 or 1
  - **Output:** equal to the input with probability $1 - p$, equal to ? with probability $p$

- **Decoding over the BEC:**
  - **Update rules:**
    - $f(a_i, a_j)$: output is ? if at least one input is ?
    - $g(a_i, a_j)$: output is ? if both inputs are ?

- **What is the erasure probability for each bit?**

---

**Density Evolution**

- **Erasure probability at $f$ nodes:** $T^f(\epsilon) = 2\epsilon - \epsilon^2$
- **Erasure probability at $g$ nodes:** $T^g(\epsilon) = \epsilon^2$
Density evolution for faulty SC decoding

• We describe failures in a memory cell as unreliable computations

Fault model
Additional erasures appear in non-erased outputs with probability $0 < \delta < 1$
Density evolution for faulty SC decoding

- We describe failures in a memory cell as unreliable computations

Fault model

Additional erasures appear in non-erased outputs with probability $0 < \delta < 1$

Density Evolution

- Erasure probability at $f$ nodes: $T^f(\epsilon) = 2\epsilon - \epsilon^2 + (1 - 2\epsilon + \epsilon^2)\delta$
- Erasure probability at $g$ nodes: $T^g(\epsilon) = \epsilon^2 + (1 - \epsilon^2)\delta$
Density evolution for faulty SC decoding

- We describe failures in a memory cell as unreliable computations

**Fault model**

Additional erasures appear in non-erased outputs with probability $0 < \delta < 1$

**Density Evolution**

- Erasure probability at $f$ nodes: $T^f(\epsilon) = 2\epsilon - \epsilon^2 + (1 - 2\epsilon + \epsilon^2)\delta$
- Erasure probability at $g$ nodes: $T^g(\epsilon) = \epsilon^2 + (1 - \epsilon^2)\delta$

- Polarization process:

$$
\epsilon_{j+1} = \begin{cases} T^f(\epsilon_j) & \text{w.p. } 1/2, \\ T^g(\epsilon_j) & \text{w.p. } 1/2, \end{cases} \quad \epsilon_0 = p.
$$
Density evolution for faulty SC decoding

- We describe failures in a memory cell as unreliable computations

Fault model

Additional erasures appear in **non-erased** outputs with probability $0 < \delta < 1$

Density Evolution

- Erasure probability at $f$ nodes: $T^f(\epsilon) = 2\epsilon - \epsilon^2 + (1 - 2\epsilon + \epsilon^2)\delta$
- Erasure probability at $g$ nodes: $T^g(\epsilon) = \epsilon^2 + (1 - \epsilon^2)\delta$

- Polarization process:

$$\epsilon_{j+1} = \begin{cases} 
T^f(\epsilon_j) & \text{w.p. } 1/2, \\
T^g(\epsilon_j) & \text{w.p. } 1/2, \\
\epsilon_0 = p. 
\end{cases}$$

Theorem (**All channels** become asymptotically **useless**)

**Under faulty SC decoding over the BEC,** $\epsilon_j \overset{a.s.}{\longrightarrow} 1.$
Improving robustness: Optimal Blocklength

- Two conflicting processes as the blocklength is increased:
  1. **Polarization** tends to decrease the FER
  2. **Internal erasures** tend to increase the FER (asymptotically dominate)
Improving robustness: Optimal Blocklength

- Two conflicting processes as the blocklength is increased:
  1. **Polarization** tends to decrease the FER
  2. **Internal erasures** tend to increase the FER (*asymptotically dominate*)
- User **upper and lower bounds** to find **FER-optimal** (finite) blocklength.
Improving robustness: Optimal Blocklength

- Two conflicting processes as the blocklength is increased:
  1. **Polarization** tends to decrease the FER
  2. **Internal erasures** tend to increase the FER (asymptotically dominate)
- User **upper and lower bounds** to find **FER-optimal** (finite) blocklength.
Error-free transmission via unequal error protection

- **Concept:** Nodes closer to the root contribute more to the erasure rate reduction
Error-free transmission via unequal error protection

- **Concept:** Nodes closer to the root contribute more to the erasure rate reduction
- Protect $n_p = (n + 1) - n_u$ levels for $n_u$ fixed: protected fraction is **constant**
Error-free transmission via unequal error protection

- **Concept:** Nodes closer to the root contribute more to the erasure rate reduction
- Protect $n_p = (n + 1) - n_u$ levels for $n_u$ fixed: protected fraction is **constant**

**Theorem (Full reliability by protecting a constant fraction of the decoder)**

1. For any fixed $n_u < n + 1$, $\epsilon_j$ converges a.s. to a random variable $\epsilon_\infty \in \{0, 1\}$. 

**Mult. penalty:** $\frac{C}{1-p}$
Error-free transmission via unequal error protection

- **Concept:** Nodes closer to the root contribute more to the erasure rate reduction
- **Protect** $n_p = (n + 1) - n_u$ levels for $n_u$ fixed: protected fraction is constant

Theorem (**Full reliability** by protecting a constant fraction of the decoder)

1. For any fixed $n_u < n + 1$, $\epsilon_j$ converges a.s. to a random variable $\epsilon_\infty \in \{0, 1\}$.
2. **Rate loss:** $P(\epsilon_\infty = 0) = \left(1 - \delta\right)^{n_u} (1 - p)$
Error-free transmission via unequal error protection

- **Concept:** Nodes closer to the root contribute more to the erasure rate reduction
- **Protect** $n_p = (n + 1) - n_u$ levels for $n_u$ fixed: protected fraction is constant

**Theorem (Full reliability by protecting a constant fraction of the decoder)**

1. For any fixed $n_u < n + 1$, $\epsilon_j$ converges a.s. to a random variable $\epsilon_\infty \in \{0, 1\}$.
2. **Rate loss:** $P(\epsilon_\infty = 0) = (1 - \delta)^{n_u} (1 - p)$

% of protected MEs:

- $n_p = 0 : 0.00\%$
- $n_p = n + 1 : 100.00\%$
Error-free transmission via unequal error protection

- **Concept:** Nodes closer to the root contribute more to the erasure rate reduction
- **Protect** $n_p = (n + 1) - n_u$ levels for $n_u$ fixed: protected fraction is **constant**

**Theorem (Full reliability by protecting a constant fraction of the decoder)**

1. For any fixed $n_u < n + 1$, $\epsilon_j$ **converges a.s.** to a random variable $\epsilon_\infty \in \{0, 1\}$.
2. **Rate loss:** $P(\epsilon_\infty = 0) = (1 - \delta)^{n_u} (1 - p)$

- **mult. penalty** \hspace{1cm} **BEC cap.**

% of protected MEs:

- $n_p = 0 : \hspace{0.5cm} 0.00\%$
- $n_p = 1 : \hspace{0.5cm} 0.05\%$
- $n_p = n + 1 : 100.00\%$
Error-free transmission via unequal error protection

- **Concept**: Nodes closer to the root contribute more to the erasure rate reduction
- Protect $n_p = (n + 1) - n_u$ levels for $n_u$ fixed: protected fraction is **constant**

Theorem (**Full reliability** by protecting a **constant fraction** of the decoder)

1. For any fixed $n_u < n + 1$, $\epsilon_j$ converges a.s. to a random variable $\epsilon_\infty \in \{0, 1\}$.
2. **Rate loss**: $P(\epsilon_\infty = 0) = (1 - \delta)^{n_u} (1 - p)$

% of protected MEs:
- $n_p = 0 : 0.00\%$
- $n_p = 1 : 0.05\%$
- $n_p = 2 : 0.15\%$
- $n_p = n + 1 : 100.00\%$
Error-free transmission via unequal error protection

- **Concept:** Nodes closer to the root contribute more to the erasure rate reduction
- **Protect** $n_p = (n + 1) - n_u$ levels for $n_u$ fixed: protected fraction is **constant**

**Theorem (Full reliability by protecting a constant fraction of the decoder)**

1. **For any fixed** $n_u < n + 1$, $\epsilon_j$ **converges a.s.** to a random variable $\epsilon_\infty \in \{0, 1\}$.
2. **Rate loss:** $P(\epsilon_\infty = 0) = (1 - \delta)^{n_u} (1 - p)$
   - *mult. penalty* BEC cap.

% of protected MEs:

- $n_p = 0$: 0.00%
- $n_p = 1$: 0.05%
- $n_p = 2$: 0.15%
- $n_p = 3$: 0.34%
- $n_p = n + 1$: 100.00%
Error-free transmission via unequal error protection

- **Concept:** Nodes closer to the root contribute more to the erasure rate reduction
- Protect $n_p = (n + 1) - n_u$ levels for $n_u$ fixed: protected fraction is constant

**Theorem (Full reliability by protecting a constant fraction of the decoder)**

1. For any fixed $n_u < n + 1$, $\epsilon_j$ converges a.s. to a random variable $\epsilon_\infty \in \{0, 1\}$.
2. **Rate loss:** $P(\epsilon_\infty = 0) = (1 - \delta)^{n_u} (1 - p)$

\[ P(\epsilon_\infty = 0) = \left(1 - \delta\right)^{n_u} \left(1 - p\right) \]

\[ \text{mult. penalty \ BEC cap.} \]

% of protected MEs:
- $n_p = 0 : 0.00\%$
- $n_p = 1 : 0.05\%$
- $n_p = 2 : 0.15\%$
- $n_p = 3 : 0.34\%$
- $n_p = 4 : 0.73\%$
- $n_p = n + 1 : 100.00\%$

![Graph showing frame erasure rate vs rate for different values of $n_p$]
Error-free transmission via unequal error protection

- **Concept**: Nodes closer to the root contribute more to the erasure rate reduction
- Protect \( n_p = (n + 1) - n_u \) levels for \( n_u \) fixed: protected fraction is constant

**Theorem** (**Full reliability** by protecting a **constant fraction** of the decoder)

1. For any fixed \( n_u < n + 1 \), \( \epsilon_j \) converges a.s. to a random variable \( \epsilon_\infty \in \{0, 1\} \).
2. **Rate loss**: \( P(\epsilon_\infty = 0) = (1 - \delta)^{n_u} (1 - p) \)

<table>
<thead>
<tr>
<th>% of protected MEs:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_p = 0 ) : 0.00%</td>
</tr>
<tr>
<td>( n_p = 1 ) : 0.05%</td>
</tr>
<tr>
<td>( n_p = 2 ) : 0.15%</td>
</tr>
<tr>
<td>( n_p = 3 ) : 0.34%</td>
</tr>
<tr>
<td>( n_p = 4 ) : 0.73%</td>
</tr>
<tr>
<td>( n_p = 5 ) : 1.51%</td>
</tr>
<tr>
<td>( n_p = n + 1 ) : 100.00%</td>
</tr>
</tbody>
</table>
Error-free transmission via unequal error protection

- **Concept:** Nodes closer to the root contribute more to the erasure rate reduction
- Protect $n_p = (n + 1) - n_u$ levels for $n_u$ fixed: protected fraction is constant

**Theorem (Full reliability by protecting a constant fraction of the decoder)**

1. For any fixed $n_u < n + 1$, $\epsilon_j$ converges a.s. to a random variable $\epsilon_\infty \in \{0, 1\}$.
2. **Rate loss:** $P(\epsilon_\infty = 0) = (1 - \delta)^{n_u} (1 - p)$

---

% of protected MEs:

- $n_p = 0 : 0.00\%$
- $n_p = 1 : 0.05\%$
- $n_p = 2 : 0.15\%$
- $n_p = 3 : 0.34\%$
- $n_p = 4 : 0.73\%$
- $n_p = 5 : 1.51\%$
- $n_p = n + 1 : 100.00\%$
Outline

1. **Classical algorithm/hardware co-design:**
   - Hardware implementation of successive cancellation list decoding of polar codes
   - Successive cancellation flip decoding of polar codes & its hardware implementation

2. **Approximate computing:**
   - Throughput-oriented construction of polar codes
   - Error-correction coding on faulty hardware

3. **Communications hardware meets information theory and machine learning:**
   - Terabit/s LDPC hardware decoders via quantized message passing
   - Neural networks for self-interference cancellation in full-duplex radios
Min-Sum Decoding of LDPC Codes

- LDPC codes are linear block codes with a sparse parity-check matrix
Min-Sum Decoding of LDPC Codes

- LDPC codes are linear block codes with a sparse parity-check matrix.

- An LDPC code can be represented as a Tanner graph with:
  - Variable nodes (VNs)
  - Check nodes (CNs)

Min-Sum Decoding

Variable-to-check messages: \( \Phi_v(L, \mu) = L + \sum_i \mu_i \),

Check-to-variable messages: \( \Phi_c(\mu) = \prod_j \text{sign} \mu_j \min |\mu| \).
Finite-Alphabet Message Passing Decoding

• In hardware implementations of MS decoding, uniform quantization is used.
Finite-Alphabet Message Passing Decoding

- In hardware implementations of MS decoding, uniform quantization is used.

Conventional Message-Passing

- **Efficient** arithmetic circuits, but **suboptimal** error-correcting performance.
Finite-Alphabet Message Passing Decoding

• In hardware implementations of MS decoding, uniform quantization is used.

Conventional Message-Passing

Decoding Algorithm $\rightarrow$ Quantization

• Efficient arithmetic circuits, but suboptimal error-correcting performance.

Finite-Alphabet Message-Passing

Quantization $\rightarrow$ Decoding Algorithm

• Updates are implemented as optimized look-up tables (LUTs).
• Potential for significant bit-width reduction and performance improvement.

Look-Up Table Design

- Our method is based on an information theoretic criterion.

**LUT Design Principle**

Local maximization of mutual information between messages and codeword bits.
Look-Up Table Design

- Our method is based on an information theoretic criterion.

LUT Design Principle

Local maximization of mutual information between messages and codeword bits.

- IEEE 802.3an LDPC code \((d_v = 6, \, d_c = 32, \, N = 2048, \, I_{\text{max}} = 5)\)

![Graph showing FER vs. Eb/No](image)
Look-Up Table Design

- Our method is based on an **information theoretic** criterion.

**LUT Design Principle**

Local **maximization of mutual information** between messages and codeword bits.

- IEEE 802.3an LDPC code \((d_v = 6, d_c = 32, N = 2048, I_{\text{max}} = 5)\)

![Graph showing FER vs. Eb/N0 for Fixed-point and Floating-point]

- Fixed-point, \((Q_{\text{msg}} = 5)\)
- Floating-point, \(I = 5\)
Look-Up Table Design

- Our method is based on an **information theoretic** criterion.

**LUT Design Principle**

Local **maximization of mutual information** between messages and codeword bits.

- IEEE 802.3an LDPC code ($d_v = 6$, $d_c = 32$, $N = 2048$, $I_{\text{max}} = 5$)
Look-Up Table Design

- Our method is based on an information theoretic criterion.

**LUT Design Principle**

Local maximization of mutual information between messages and codeword bits.

- IEEE 802.3an LDPC code \( (d_v = 6, \ d_c = 32, \ N = 2048, \ I_{\text{max}} = 5) \)

![Graph showing FER vs. Eb/N0 for different message bit-widths and representations.](image-url)
Look-Up Table Design

- Our method is based on an **information theoretic** criterion.

**LUT Design Principle**

**Local maximization of mutual information** between messages and codeword bits.

- IEEE 802.3an LDPC code \((d_v = 6, d_c = 32, N = 2048, I_{\text{max}} = 5)\)

![Graph showing FER vs. Eb/N0 for different Qmsg values.]

40% message bit-width reduction with identical FER performance.
Fully Unrolled LDPC Decoder Hardware Architecture

- One CN stage and one VN stage per iteration: $2I_{\text{max}}$ pipeline stages.
Fully Unrolled LDPC Decoder Hardware Architecture

- One CN stage and one VN stage per iteration: $2I_{\text{max}}$ pipeline stages.

<table>
<thead>
<tr>
<th></th>
<th>Quant. MS</th>
<th>LUT-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area (mm$^2$)</td>
<td>35.63</td>
<td>33.79</td>
</tr>
<tr>
<td>Throughput (Gbps)</td>
<td>1014</td>
<td>1665</td>
</tr>
<tr>
<td>Area Eff. (Gbps/mm$^2$)</td>
<td>28.46</td>
<td>49.27</td>
</tr>
</tbody>
</table>

- Area reduction: $-5\%$
- Throughput increase: $+64\%$
- Area efficiency increase: $+73\%$
Bi-directional Wireless Communications

Time-division duplexing (TDD)
Wasted time resources: switching interval

Frequency-division duplexing (FDD)
Wasted frequency resources: guard bands

In-Band Full-duplex (IBFD)
Up to twice the throughput wrt TDD & FDD!
No additional bandwidth
No wasted time or frequency resources
Bi-directional Wireless Communications

Time-division duplexing (TDD)
Wasted time resources: switching interval

Frequency-division duplexing (FDD)
Wasted frequency resources: guard bands

In-Band Full-duplex (IBFD)
Up to twice the throughput wrt TDD & FDD!
No additional bandwidth
No wasted time or frequency resources

Fundamental Challenge: Self-interference is much stronger than the desired signal!
Why is Cancellation Challenging?

- **In principle**, cancellation is easy since digital transmitted signal is known!
Why is Cancellation Challenging?

- **In principle**, cancellation is easy since digital transmitted signal is known!
- **In practice**, the digital signal does not tell the whole story!
Why is Cancellation Challenging?

• **In principle**, cancellation is easy since digital transmitted signal is known!
• **In practice**, the digital signal does not tell the whole story!
  - Analog components introduce non-linearities that make digital cancellation difficult.
Why is Cancellation Challenging?

- **In principle**, cancellation is easy since digital transmitted signal is known!
- **In practice**, the digital signal does not tell the whole story!
  - Analog components introduce non-linearities that make **digital cancellation difficult**
- Consider a state-of-the-art non-linear cancellation model:

\[
y(n) = \sum_{p=1}^{P} \sum_{q=0}^{p} \sum_{m=0}^{M+L-1} h_{p,q}(m) x(n-m)^q x^*(n-m)^{p-q}
\]

**Example**

For \( P = 7 \) and \( M + L = 13 \) memory taps \( \rightarrow 20 \) basis functions and 260 parameters!

---

Why is Cancellation Challenging?

- **In principle**, cancellation is easy since digital transmitted signal is known!
- **In practice**, the digital signal does not tell the whole story!
  - Analog components introduce non-linearities that make digital cancellation difficult
- Consider a state-of-the-art non-linear cancellation model:

  \[
  y(n) = \sum_{p=1}^{P} \sum_{q=0}^{p} \sum_{m=0}^{M+L-1} h_{p,q}(m) x(n-m)^q x^*(n-m)^{p-q}
  \]

  **Example**

  For \( P = 7 \) and \( M + L = 13 \) memory taps \( \rightarrow \) 20 basis functions and 260 parameters!

  **Idea**

  Why not use a **neural network** that extracts structure from training data?

---


Self-Interference Cancellation Using Neural Networks

- **Decompose** self-interference signal into linear and non-linear part

\[
y(n) = y_{\text{lin}}(n) + y_{\text{nl}}(n)
\]

- \(y_{\text{lin}}(n)\) is easy!
- \(y_{\text{nl}}(n)\) is hard!
Self-Interference Cancellation Using Neural Networks

- **Decompose** self-interference signal into linear and non-linear part

\[ y(n) = y_{\text{lin}}(n) + y_{\text{nl}}(n) \]

  - easy!
  - hard!

- **Two-step cancellation:**
  1. Use standard linear digital cancellation:
     \[ \hat{y}_{\text{lin}}(n) = \sum_{m=0}^{M+L-1} \hat{h}_{1,1}(m)x(n - m) \]
Self-Interference Cancellation Using Neural Networks

- **Decompose** self-interference signal into linear and non-linear part

\[ y(n) = y_{\text{lin}}(n) + y_{\text{nl}}(n) \]

- **Two-step cancellation:**
  1. Use standard linear digital cancellation:  
     \[ \hat{y}_{\text{lin}}(n) = \sum_{m=0}^{M+L-1} \hat{h}_{1,1}(m)x(n-m) \]
  2. Train a neural network to reproduce and cancel  
     \[ y_{\text{nl}}(n) \approx y(n) - \hat{y}_{\text{lin}}(n) \]
Experimental Cancellation Results

- 10 MHz OFDM signal, 56 dB passive cancellation, $M + L = 13$ taps
Experimental Cancellation Results

- 10 MHz OFDM signal, 56 dB passive cancellation, $M + L = 13$ taps
Experimental Cancellation Results

- 10 MHz OFDM signal, 56 dB passive cancellation, $M + L = 13$ taps
Experimental Cancellation Results

- 10 MHz OFDM signal, 56 dB passive cancellation, $M + L = 13$ taps
Experimental Cancellation Results

- 10 MHz OFDM signal, 56 dB passive cancellation, $M + L = 13$ taps
Experimental Cancellation Results

• 10 MHz OFDM signal, 56 dB passive cancellation, $M + L = 13$ taps

![Graph showing power spectral density vs frequency for different linear and non-linear methods.]

<table>
<thead>
<tr>
<th></th>
<th>Poly.</th>
<th>NN</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additions</td>
<td>492</td>
<td>493</td>
<td>0%</td>
</tr>
<tr>
<td>Multiplications</td>
<td>741</td>
<td>476</td>
<td>36%</td>
</tr>
</tbody>
</table>
Experimental Cancellation Results

- 10 MHz OFDM signal, 56 dB passive cancellation, $M + L = 13$ taps

**Identical** cancellation performance with **lower complexity**!
A combination of approaches is necessary to maintain progress in the **post-happy-scaling era**, including:

1. Classical algorithm/hardware co-design
2. Approximate computing
3. Tools from other disciplines
Conclusions

A combination of approaches is necessary to maintain progress in the post-happy-scaling era, including:
1. Classical algorithm/hardware co-design
2. Approximate computing
3. Tools from other disciplines

The end of the happy scaling era creates new challenges and opportunities for innovation!