Polar FEC Codes Running at Hundreds of Mbit/s in GNU Radio or On a Software Implementation of the Fast-SSC Algorithm

Pascal Giard\textsuperscript{1} and Alexandre J. Raymond\textsuperscript{2}

\textsuperscript{1}McGill University, Department of Electrical and Computer Engineering, Montréal, Québec, Canada
pascal.giard@mail.mcgill.ca

\textsuperscript{2}École de technologie supérieure Montréal, Québec, Canada
alexandre.raymond@lacime.etsmtl.ca

August 26, 2015
Who am I?

- Ph.D. candidate and research assistant at McGill University.
- Lecturer and research professional at ÉTS.
- Work on efficient algorithms and implementations for modern error-correction codes with a special focus on polar codes.
  - FPGA and ASIC
  - Intel/AMD x86-64 and ARM NEON SIMD
  - GPGPU
Introduction to Error Correction

P. Giard and A. J. Raymond (McGill and ETS)  Polar Codes in GNU Radio
Introduction to Error Correction

Cat

→

Parity Codeword Decoder

P. Giard and A. J. Raymond (McGill and ETS)
Introduction to Error Correction

![Diagram showing a cat and a bat in conversation about parity codewords and decoders.]
Introduction to Error Correction

Cat, meow

Bat, meow

P. Giard and A. J. Raymond (McGill and ETS)  Polar Codes in GNU Radio
Introduction to Error Correction

Cat, meow

Bat, meow
→ Cat
Introduction to Error Correction

Codeword

Parity

Cat, meow

→

Bat, meow

→ Cat

Decoder
Introduction to Error Correction

How good can we make this in a digital system?

P. Giard and A. J. Raymond (McGill and ETS)  Polar Codes in GNU Radio
Achieving the Channel Capacity

Definition

1948

LDPC Codes

1960

Turbo Codes

1993

0.7 dB

LDPC Codes

2004

0.04 dB

Polar Codes

2008

0 dB

using SC

as $N \to \infty$
Achieving the Channel Capacity

- **Definition**: 1948
- **LDPC Codes**: 1960
- **Turbo Codes**: 1993 (0.7 dB)
- **LDPC Codes**: 2004 (0.04 dB)
- **Polar Codes**: 2008 (0 dB using SC as $N \to \infty$)

Very challenging in software.

P. Giard and A. J. Raymond (McGill and ETS)  
Polar Codes in GNU Radio
Achieving the Channel Capacity

Successive cancellation $\rightarrow$ low throughput.

$\Rightarrow$ Use a better algorithm.

Performance at moderate code-length is modest.
Becomes excellent with $N \geq 2^{20}$.

$\Rightarrow$ Still competitive under some conditions;
use long frames if required.
Fast-SSC Decoding

What better algorithm? Fast-SSC!

- Orders of magnitude more efficient than SC;
- Maps nicely into SIMD;
- Memory can be arranged in an efficient way;
- Natural fit for systematic coding.

Wait! What is a systematic code?

Non-systematic and Systematic Coding

(8, 4) code → 8-bit codeword: 4 information bits + 4 parity bits.

- Non-systematic:
  \[ a_0 a_1 a_2 a_3 \rightarrow \text{Encoder} \rightarrow x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 \]

- Systematic:
  \[ a_0 a_1 a_2 a_3 \rightarrow \text{Encoder} \rightarrow a_0 a_1 a_2 a_3 p_0 p_1 p_2 p_3 \]
Non-systematic Polar Encoding

Figure: Structure for a (8, 4) polar code.
Systematic Polar Encoding

Figure: Low-complexity structure for a (8, 4) polar code.

Alright, now how do we go from SC to Fast-SSC decoding?

Successive-Cancellation Decoding

- Reminder of how SC decoding works.
Successive-Cancellation Decoding

- Reminder of how SC decoding works.
Successive-Cancellation Tree

- View the SC decoder graph as a tree.

\[ \hat{u}_0 + \hat{u}_1 + \hat{u}_2 + \hat{u}_3 + \hat{u}_4 + \hat{u}_5 + \hat{u}_6 + \hat{u}_7 = y_0 \]
\[ \hat{u}_1 + \hat{u}_2 + \hat{u}_3 + \hat{u}_4 + \hat{u}_5 + \hat{u}_6 + \hat{u}_7 = y_1 \]
\[ \hat{u}_2 + \hat{u}_3 + \hat{u}_4 + \hat{u}_5 + \hat{u}_6 + \hat{u}_7 = y_2 \]
\[ \hat{u}_3 + \hat{u}_4 + \hat{u}_5 + \hat{u}_6 + \hat{u}_7 = y_3 \]
\[ \hat{u}_4 + \hat{u}_5 + \hat{u}_6 + \hat{u}_7 = y_4 \]
\[ \hat{u}_5 + \hat{u}_6 + \hat{u}_7 = y_5 \]
\[ \hat{u}_6 + \hat{u}_7 = y_6 \]
\[ \hat{u}_7 = y_7 \]
Successive-Cancellation Tree

- View the SC decoder graph as a tree.
View the SC decoder graph as a tree.
Successive-Cancellation Tree

View the SC decoder graph as a tree.
Successive-Cancellation Tree

- View the SC decoder graph as a tree.
Successive-Cancellation Tree

- View the SC decoder graph as a tree.
Simplified Successive Cancellation


P. Giard and A. J. Raymond (McGill and ETS)  Polar Codes in GNU Radio
Key idea: use more efficient algorithms on constituent codes.

Fast-SSC supports more node types not covered here.

SC Tree—(256, 230)
SSC and Fast-SSC Trees—(256, 230)

SSC

Fast-SSC
Implementation for GNU Radio

- Systematic polar coding for better BER.
- Supports a subset of the Fast-SSC nodes.
- Uses the FEC API.
- Supports soft inputs for greater error-correction performance.
- Uses fixed-point arithmetic internally.
- Highly-optimized VOLK kernels for SSE4, AVX2 and NEON.
Performance Results

Using a single processor core:

- Under the worst conditions, a stable 50 Mbps on a low-power SandyBridge with SSE4.1.
- Under good conditions, an average coded throughput of 350 Mbps on a Haswell with AVX2 is typical.

⇒ In a complete communication chain, the modern FEC is NOT the bottleneck anymore.
Performance Results Example

Polar FEC Encoder/Decoder

(encoder implementation not optimal)
Final Words

- Will this be upstreamed to GNU Radio/VOLK?
- Will we make the source code available?
- When can you start using it?
- What if you want to use polar codes outside GNU Radio?
Acknowledgement

The authors would like to warmly thank:

- Profs. Ghyslain Gagnon and Claude Thibeault of ETS for funding this trip.
- Prof. François Gagnon of ETS for HW access and helpful discussions.
- Gabi Sarkis of McGill university for helpful discussions and presentation material.

Contact information

- Pascal.Giard@mail.mcgill.ca
- Alexandre.Raymond@lacime.etsmtl.ca