Beyond Gbps Turbo Decoder on Multi-Core CPUs

Adrien Cassagne, Thibaud Tonnellier, Camille Leroux,
Bertrand Le Gal, Olivier Aumage and Denis Barthou

Software Implementations of FEC decoders

Growing interest for Software Defined Radio (SDR)

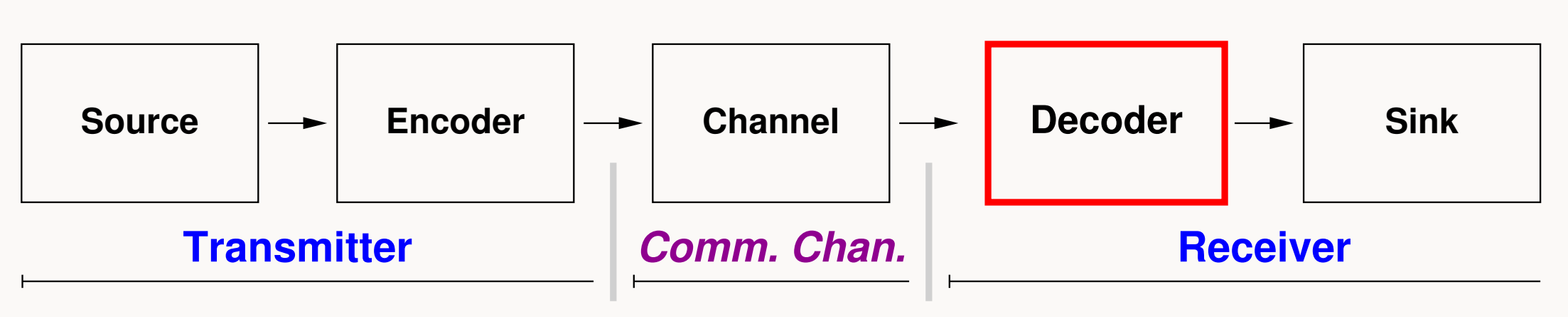

Figure 1: Simplified communication chain

Leverage powerful, energy efficient procs (x86, ARM):

a. Validate and optimize new algorithms
b. Implement real systems with real time performance
   - Enable Cloud computing-based architecture for Radio Access Networks (C-RAN)
   - Reduce dev. cost and time to market

$\rightarrow$ Need for efficient software implementations to limit the energy consumption with high throughput.

Turbo-coding / decoding

A turbo code is a parallel concatenation of two component convolutional codes.
The turbo-decoding process is an iterative method in which two soft-input soft-output (SISO) decoders exchange extrinsic information via an interleaver.

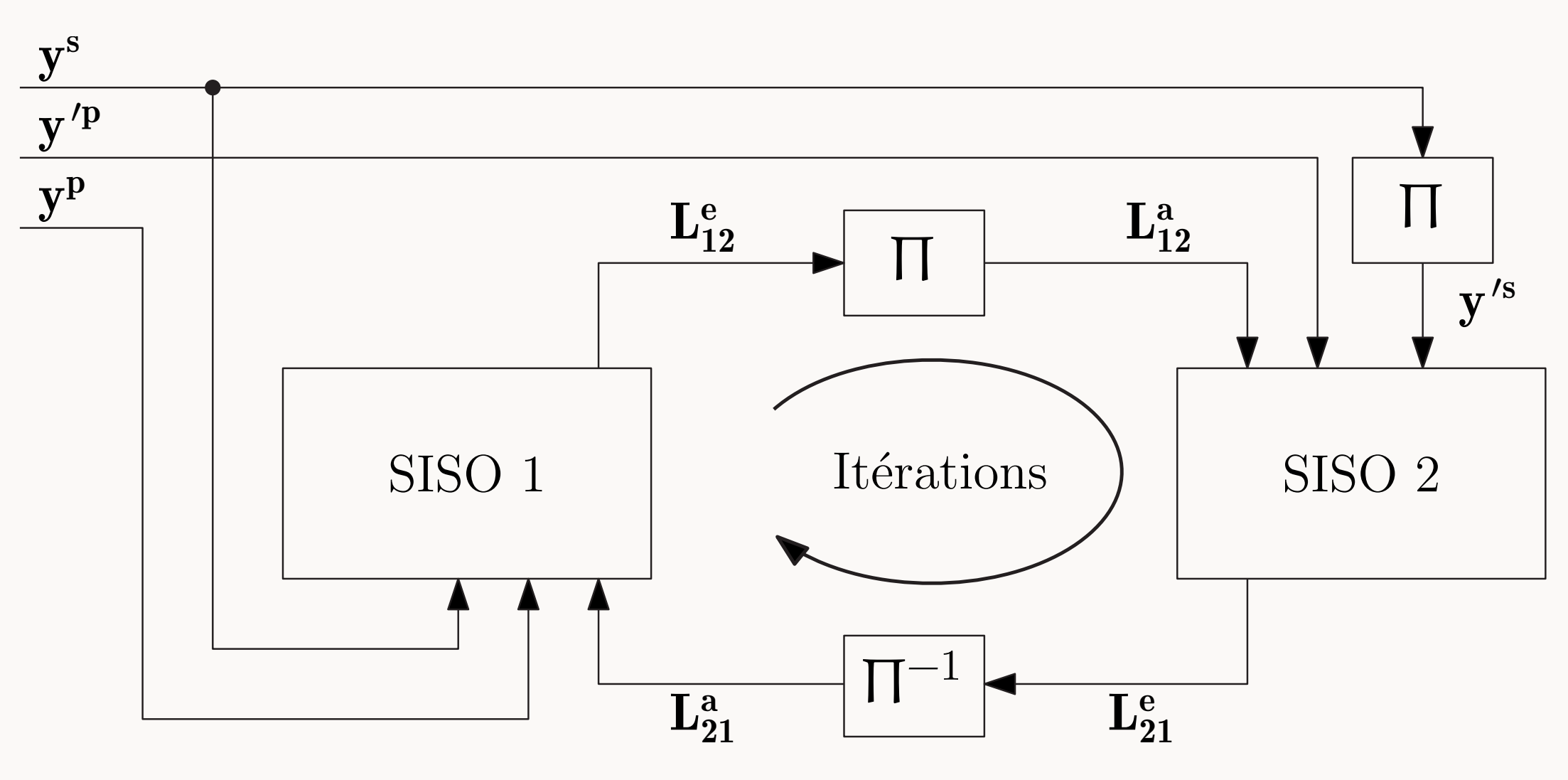

Figure 2: Turbo-decoding process

Loop merging in BCJR implementation

Standard BCJR implementation:

```
1: for all frames do                                          ▷ Sequential loop
2:     for $k = 0;\ k < K;\ k = k + 1$ do                    ▷ Parallel loop
3:         $\gamma^k \leftarrow computeGamma(L^k_{sys}, L^k_p, L^k_e)$
4:     $\alpha^0 \leftarrow initAlpha()$
5:     for $k = 1;\ k < K;\ k = k + 1$ do                    ▷ Sequential loop
6:         $\alpha^k \leftarrow computeAlpha(\alpha^{k-1}, \gamma^{k-1})$
7:     $\beta^{K-1} \leftarrow initBeta()$
8:     for $k = K - 2;\ k \geq 0;\ k = k - 1$ do             ▷ Sequential loop
9:         $\beta^k \leftarrow computeBeta(\beta^{k+1}, \gamma^k)$
10:    for $k = 0;\ k < K;\ k = k + 1$ do                    ▷ Parallel loop
11:        $L^k_e \leftarrow computeExtrinsic(\alpha^k, \beta^k, \gamma^k)$
```

Loop merging BCJR implementation

```
1: for all frames do                                          ▷ Vectorized loop
2:     $\alpha^0 \leftarrow initAlpha()$
3:     for $k = 1;\ k < K;\ k = k + 1$ do                    ▷ Sequential loop
4:         $\gamma^{k-1} \leftarrow computeGamma(L^{k-1}_{sys}, L^{k-1}_p, L^{k-1}_e)$
5:         $\alpha^k \leftarrow computeAlpha(\alpha^{k-1}, \gamma^{k-1})$
6:     $\gamma^{K-1} \leftarrow computeGamma(L^{K-1}_{sys}, L^{K-1}_p, L^{K-1}_e)$
7:     $\beta^{K-1} \leftarrow initBeta()$
8:     $L^{K-1}_e \leftarrow computeExtrinsic(\alpha^{K-1}, \beta^{K-1}, \gamma^{K-1})$
9:     for $k = K - 2;\ k \geq 0;\ k = k - 1$ do             ▷ Sequential loop
10:        $\beta^k \leftarrow computeBeta(\beta^{k+1}, \gamma^k)$
11:        $L^k_e \leftarrow computeExtrinsic(\alpha^k, \beta^k, \gamma^k)$
```

References

[1] M. Wu, Y. Sun, and J. R. Cavallaro. Implementation of a 3GPP LTE turbo decoder accelerator on GPU. In *IEEE SiPS*, 2010.
[2] Michael Wu, Yang Sun, Guohui Wang, and Joseph R. Cavallaro. Implementation of a high throughput 3GPP turbo decoder on GPU. *Springer JSPS*, 65(2):171–183, 2011.
[3] S. Chinnici and P. Spallaccini. Fast simulation of turbo codes on GPUs. In *IEEE ISTC*, pages 61–65, 2012.
[4] D.R.N. Yoge and N. Chandrachoodan. GPU implementation of a programmable turbo decoder for software defined radio applications. In *IEEE VLSID*, 2012.
[5] Chengjun Liu, Zhisong Bie, Canfeng Chen, and Xianjun Jiao. A parallel LTE turbo decoder on GPU. In *IEEE ICCT*, 2013.
[6] Xiang Chen, Ji Zhu, Ziyu Wen, Yu Wang, and Huazhong Yang. BER guaranteed optimization and implementation of parallel turbo decoding on GPU. In *IEEE ICST*, 2013.
[7] Jiao Xianjun, Chen Canfeng, P. Jaaskelainen, V. Guzma, and H. Berg. A 122mb/s turbo decoder using a mid-range GPU. In *IEEE IWCMC*, 2013.
[8] Michael Wu, Guohui Wang, Bei Yin, Christoph Studer, and Joseph R. Cavallaro. HSPA+/LTE-A turbo decoder on GPU and multicore CPU. In *IEEE ACSSC*, 2013.
[9] Yang Zhang et al. The acceleration of turbo decoder on the newest GPGPU of kepler architecture. In *IEEE ISCIT*, 2014.
[10] Rongchun Li, Yong Dou, Jiaqing Xu, Xin Niu, and Shice Ni. An efficient parallel SOVA-based turbo decoder for software defined radio on GPU. *IEICE Trans. Fundamentals*, 97(5):1027–1036, 2014.
[11] Lin Huang et al. A high speed turbo decoder implementation for CPU-based SDR system. In *IEEE IET ICCTA*, 2011.
[12] Suiping Zhang, Rongrong Qian, Tao Peng, Ran Duan, and Kuilin Chen. High throughput turbo decoder design for GPP platform. In *IEEE ICST*, 2012.

Contribution of this Work

Two categories of parallelism in Turbo-decoding:

- **Intra frame parallelism** : a single codeword is processed at the time. Computation within the turbo-decoding process are parallelized (trellis transitions, BCJR computations, sub-blocks, ...)
- **Inter frame parallelism** : several frames are processed at the same time. This increases latency but allows more regular memory accesses.

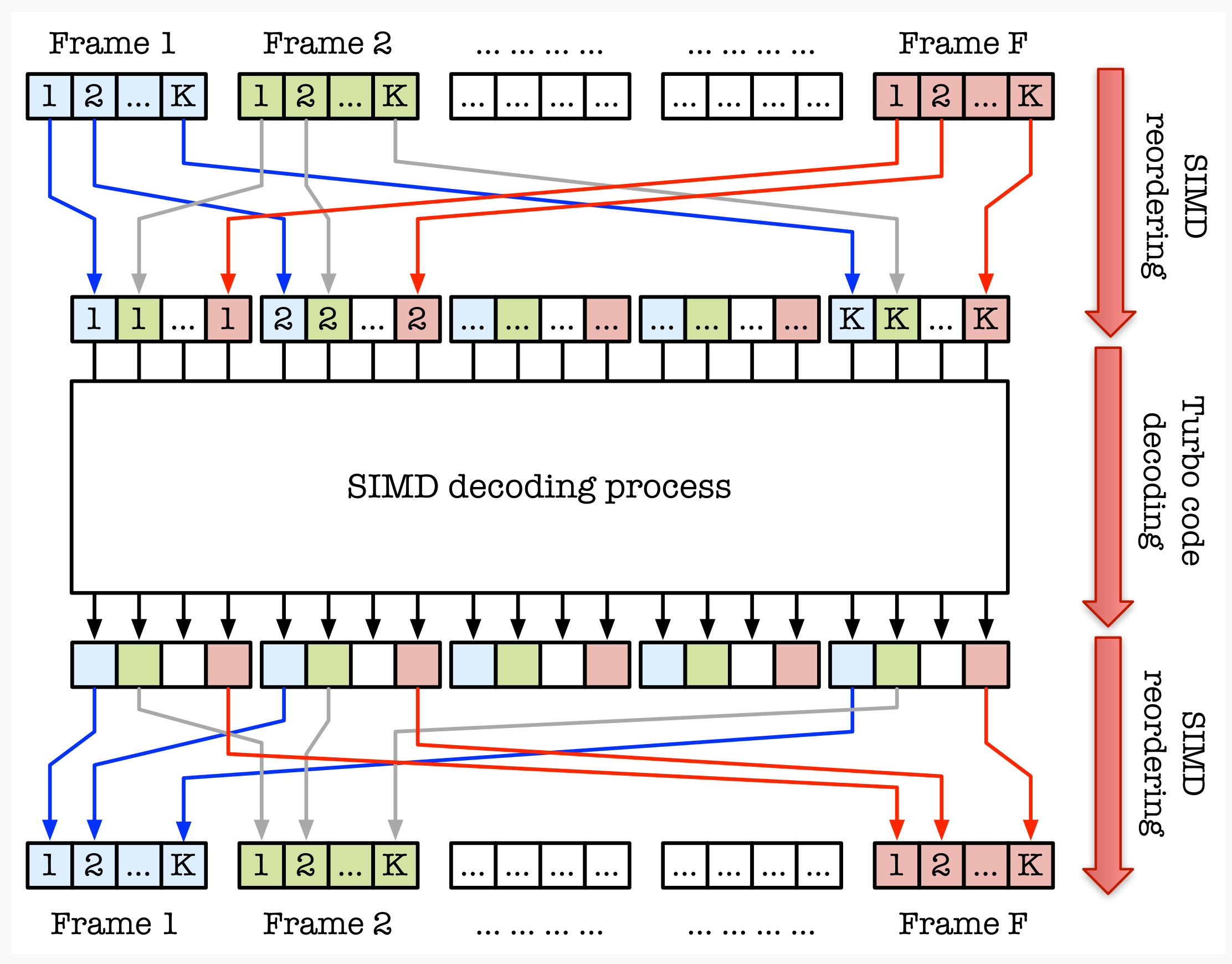

Figure 3: Inter frame parallelism mapped on SIMD units

State of the art implementations:

- Dedicated Hardware:
  - low energy, high throughput, low latency, low flexibility.
  - Intra frame parallelism is usually exploited.
  - Inter frame is inefficient since it requires the duplication of turbo-decoders.
- GPU:
  - high energy, moderate throughput, high latency, high flexibility.
  - Intra frame strategy can be used.
  - Inter frame allows regular data access.
  - 10x slower and 100x more power consuming than HW implementation.
- CPU:
  - high energy, moderate throughput, high latency, high flexibility.
  - Intra frame strategy can be used (similar to dedicated hardware)
  - Inter frame strategy can be used (not reported to date)

Contribution :

In this work, we propose a **generic and flexible CPU implementation of a turbo decoder** that exclusively uses **inter-frame parallelism**. Experimental results show that our turbo decoder outperforms existing implementations in terms of throughput and energy efficiency.

The software platform: A Fast Forward Error Correction Tool (AFF3CT)

AFF3CT: a software dedicated to simulations of digital communications with channel coding

`http://aff3ct.github.io`

- Support different coding scheme: **Polar**, **Turbo**, **Convolutional**, **Repeat and Accumulate** and **LDPC** (coming soon)
- Very fast simulations, take advantage of today CPUs architecture (**hundreds of Mb/s on Intel Core i5/7**)
  - Written in `C++11` (**SystemC/TLM support**)
  - Monte-Carlo **multi-threaded** simulations
  - Upto **1000 times faster than MATLAB** code
- **Portable**: run on Linux, Mac OS X and Windows
- **Open-source code** (under MIT license)
- For Turbo coding simulations, the following items are configurable: generator polynomial, interleaver, SISO heuristic (log-MAP, max-log-MAP,...), puncturing, trellis termination.

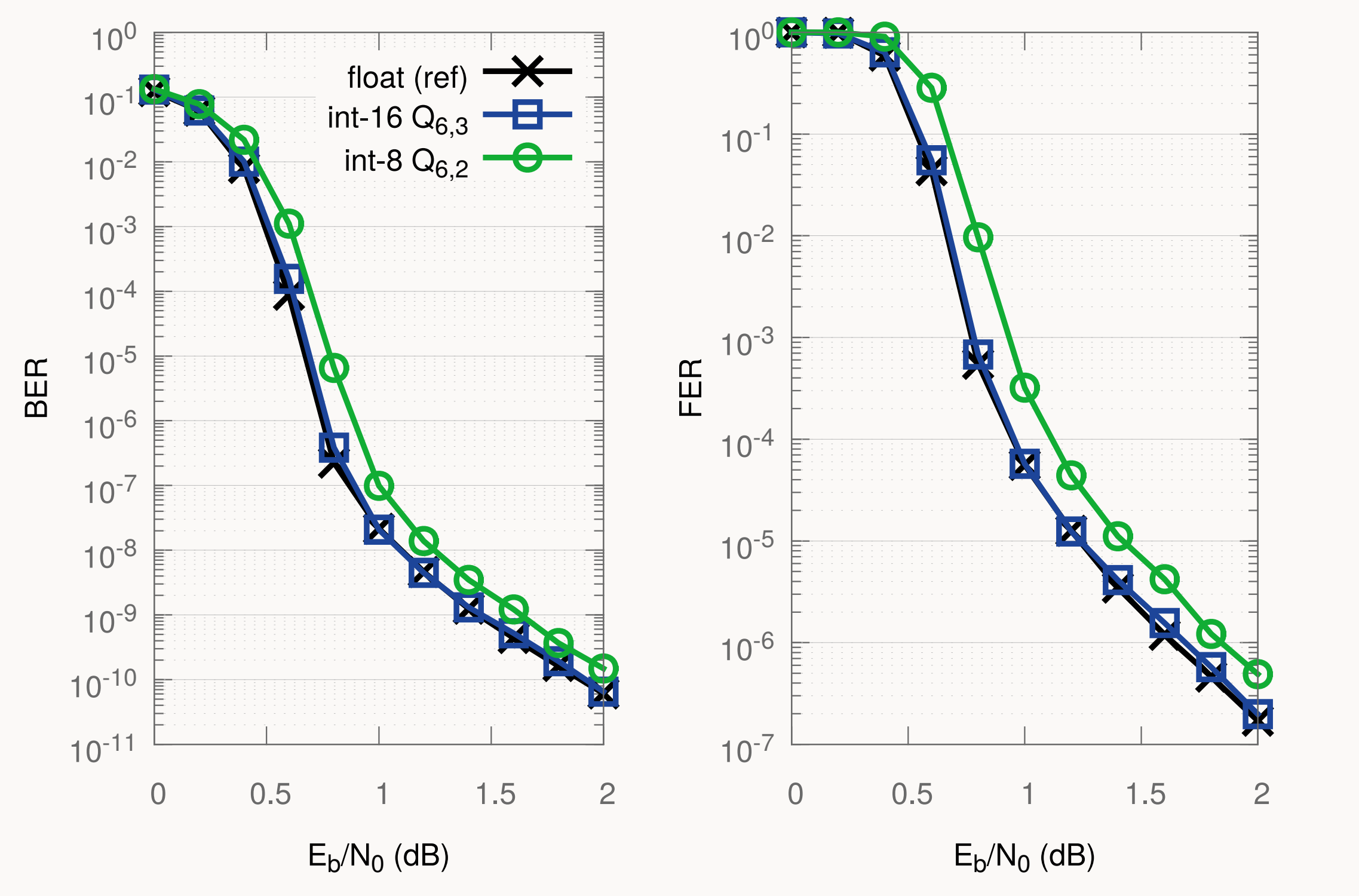

Figure 4: Simulated BER and FER of the $K = 6144$, $R = 1/3$ LTE Turbo code with AFF3CT

Experiments and Measurements

	Work	Year	Platform	Arch.	TDP	Cores	Freq.	Algorithm	Pre.	SIMD	Inter	K	Iters	BER	FER	Lat.	Thr.	NThr.	TNDC	E_d
			Hardware and decoder parameters											**Decoding performances**				**Metrics**		
					Watts	or SM	GHz		bit	length	level			at 0.7 dB		μs	Mbps	Mbps		nJ
GPU-based	[1]	2010	Tesla C1060	*Tesla*	200	15	1.30	ML-MAP	32	16	100	6144	5	1e-04	–	76800	8.0	2.1	0.135	5000
	[2]	2011	GTX 470	*Fermi*	215	14	1.22	ML-MAP	32	32	100	6144	5	4e-05	–	20827	29.5	8.6	0.270	1458
	[3]	2012	Tesla C2050	*Fermi*	247	14	1.15	L-MAP	32	32	32	11918	5	–	–	108965	3.5	1.1	0.035	14114
	[4]	2012	9800 GX2	*Tesla*	197	16	1.50	ML-MAP	32	16	1	6144	5	1e-02	–	3072	2.0	0.4	0.025	19700
	[5]	2013	GTX 550 Ti	*Fermi*	116	6	1.80	EML-MAP	32	32	1	6144	6	1e-02	–	72^*	85.3	47.4	1.482	227
	[6]	2013	GTX 580	*Fermi*	244	16	1.54	ML-MAP	32	32	1	6144	6	3e-04	–	1660	3.7	0.9	0.030	10090
	[7]	2013	GTX 480	*Fermi*	250	15	1.40	EML-MAP	32	32	1	6144	6	–	–	50^*	122.8	35.1	1.098	339
	[8]	2013	GTX 680	*Kepler*	195	8	1.01	EML-MAP	32	192	16	6144	6	–	1e-02	2657	37.0	27.5	0.144	878
	[9]	2014	Tesla K20c	*Kepler*	225	13	0.71	ML-MAP	32	192	1	6144	5	1e-04	–	1097	5.6	3.0	0.015	8036
	[10]	2014	GTX 580	*Fermi*	244	16	1.54	BR-SOVA	8	32	4	6144	5	2e-02	–	192^*	127.8	25.9	0.810	382
CPU-based	[11]	2011	i7-960	*Nehalem*	130	1	3.20	ML-MAP	16	8	1	1008	8	3e-03	7e-02	138	7.3	18.3	2.280	2226
	[12]	2012	X5670	*Westmere*	95	6	2.93	EML-MAP	8	16	6	5824	3	6e-02	–	157	222.6	38.0	2.373	142
	[8]	2013	i7-3770K	*Ivy Bridge*	77	4	3.50	EML-MAP	16	8	4	6144	6	–	1e-01	323	76.2	32.7	4.080	168
	this work	2016	E5-2650	*Ivy Bridge*	95	8	2.50	EML-MAP	16	8	64	6144	6	6e-06	6e-03	3665	107.3	32.2	4.014	148
			i7-4960HQ	*Haswell*	47	4	3.20				32					2212	88.9	41.7	5.208	88
			2×E5-2680v3	*Haswell*	240	24	2.50				192					2657	443.7	44.4	5.544	90
			E5-2650	*Ivy Bridge*	95	8	2.50		8	16	128			8e-05	5e-02	3492	225.2	67.6	4.224	70
			i7-4960HQ	*Haswell*	47	4	3.20				64					2837	138.6	65.0	4.062	57
			2×E5-2680v3	*Haswell*	240	24	2.50				384					3293	716.4	71.6	4.476	56

Conclusion

In this work a generic and flexible CPU implementation of a turbo decoder that exclusively uses inter-frame parallelism. Experimental results show that our turbo decoder outperforms existing implementations in terms of throughput and energy efficiency.

Acknowledgements

This study has been carried out with financial support from the French State, managed by the French National Research Agency (ANR) in the frame of the "Investments for the future" Programme IdEx Bordeaux - CPU (**ANR-10-IDEX-03-02**).