Energy Consumption Analysis of Software Polar Decoders on Low Power Processors

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

▶ **To cite this version:**

Adrien Cassagne, Olivier Aumage, Camille Leroux, Denis Barthou, Bertrand Le Gal. Energy Consumption Analysis of Software Polar Decoders on Low Power Processors. The 24nd European Signal Processing Conference (EUSIPCO 2016), Aug 2016, Budapest, Hungary. hal-01363975

HAL Id: hal-01363975

https://hal.archives-ouvertes.fr/hal-01363975

Submitted on 15 Nov 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Energy Consumption Analysis of Software Polar Decoders on Low Power Processors

Adrien Cassagne*†, Olivier Aumage†, Camille Leroux*, Denis Barthou† and Bertrand Le Gal*
*IMS Lab, Bordeaux INP, France
†Inria / Labri, Univ. Bordeaux, INP, France

***Abstract*—This paper presents a new dynamic and fully generic implementation of a Successive Cancellation (SC) decoder (multi-precision support and intra-/inter-frame strategy support). This fully generic SC decoder is used to perform comparisons of the different configurations in terms of throughput, latency and energy consumption. A special emphasis is given on the energy consumption on low power embedded processors for software defined radio (SDR) systems. A N=4096 code length, rate 1/2 software SC decoder consumes only 14 nJ per bit on an ARM Cortex-A57 core, while achieving 65 Mbps. Some design guidelines are given in order to adapt the configuration to the application context.**

I. Introduction

Channel coding enables transmitting data over unreliable communication channels. While error correction coding/decoding is usually performed by dedicated hardware circuits on communication devices, the evolution of general purpose processors in terms of energy efficiency and parallelism (vector processing, number of cores,...) drives a growing interest for software ECC implementations (e.g. LDPC decoders [1]–[3], *Turbo* decoders [4], [5]). The family of the *Polar* codes has been introduced recently. They asymptotically reach the capacity of various communication channels [6]. They can be decoded using a successive cancellation (SC) decoder, which has extensively been implemented in hardware [7]–[13]. Several software decoders have also been proposed [14]–[19], all employing Single Instruction Multiple Data (SIMD) instructions to reach multi-Gb/s performance. Two SIMD strategies deliver high performance: the *intra*-frame parallelism strategy [14]–[16] delivers both high throughput and low latency; the *inter*-frame parallelism strategy [17], [18] improves the throughput performance by a better use of the SIMD unit width at the expense of a higher latency. AFF3CT[1] [19], [20] (previously called P-EDGE) is the first software SC decoder to include both parallelism strategies as well as state-of-the-art throughput and latency.

The optimization space exploration for SC decoding of Polar codes has so far primarily been conducted with raw performance in mind. However, the energy consumption minimization should also be factored in. Moreover, heterogeneous multi-core processors such as ARM's big.LITTLE architectures offer cores with widely different performance and energy consumption profiles, further increasing the number of design and run-time options. In this context, the contribution of this paper is to propose a new dynamic SC decoder, integrated into our AFF3CT software and to derive key guidelines and general strategies in balancing performance and energy consumption characteristics of software SC decoders.

[1]AFF3CT is an Open-source software (MIT license) for fast forward error correction simulations, see `http://aff3ct.github.io`

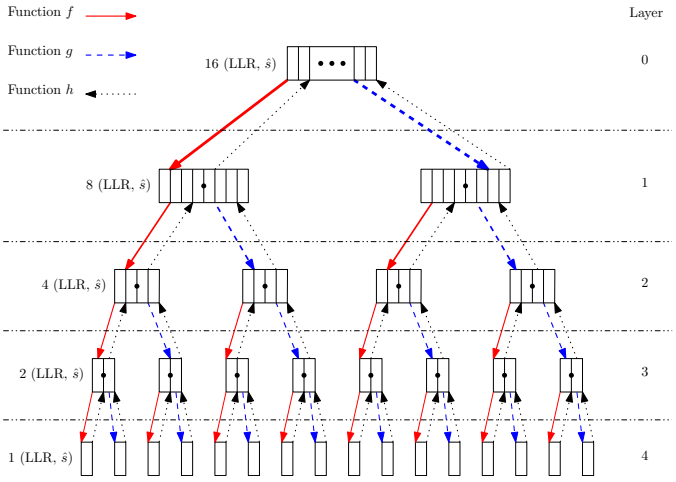

Fig. 1. Full SC decoding tree ($N = 16$)

The remainder of this paper is organized as follows. Section II details relevant characteristics of the general Polar code encoding/decoding process. Section III discusses related works in the domain. Section IV describes our proposed dynamic SC decoder and compares it to our previous *specialized* approach based on code generation. Section V presents various characteristics to explore in order to reach a performance trade-off. Section VI presents experiments and comments on performance results.

II. Polar Codes Encoding and Decoding

Polar codes are linear block codes of size $N = 2^n$, $n \in \mathbb{N}$. In [6], Arıkan defined their construction based on the n^{th} Kronecker power of a kernel matrix $\kappa = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, denoted $\kappa^{\otimes n}$. The systematic encoding process [21] consists in building an N-bit vector V including K information bits and $N - K$ frozen bits, usually set to zero. The location of the frozen bits depends on both the type of channel that is considered and the noise power on the channel [6]. Then, a first encoding phase is performed: $U = V * \kappa^{\otimes n}$ and bits of U in the frozen location are replaced by zeros. The codeword is finally obtained with a second encoding phase: $X = U * \kappa^{\otimes n}$.

In this systematic form X includes K information bits and $N-K$ redundancy bits located on the frozen locations.

After being sent over the transmission channel, the noisy version of the codeword X is received as a log likelihood ratio (LLR) vector Y. The SC decoder successively estimates each bit u_i based on the vector Y and the previously estimated bits ($[\hat{u}_0 ... \hat{u}_{i-1}]$). To estimate each bit u_i, the decoder computes the following LLR value:

$$\lambda_i^0 = \log \frac{\Pr(Y, \hat{u}_{0:i-1} | u_i = 0)}{\Pr(Y, \hat{u}_{0:i-1} | u_i = 1)}.$$

The estimated bit $\hat{u}_i$ is 0 if $\lambda_i^0 > 0$, 1 otherwise. Since the decoder knows the location of the frozen bits, if u_i is a frozen bit, $\hat{u}_i = 0$ regardless of λ_i^0 value. The SC decoding process can be seen as the traversal of a binary tree as shown in Figure 1. The tree includes $\log N + 1$ layers each including 2^d nodes, where d is the depth of the layer in the tree. Each node contains a set of 2^{n-d} LLRs and partial sums $\hat{s}$. Nodes are visited using a pre-order traversal. As shown in Figure 1, three functions, f, g and h are used for node updates:

$$\begin{cases} f(\lambda_a, \lambda_b) &= sign(\lambda_a.\lambda_b).\min(|\lambda_a|, |\lambda_b|) \\ g(\lambda_a, \lambda_b, s) &= (1-2s)\lambda_a + \lambda_b \\ h(s_a, s_b) &= (s_a \oplus s_b, s_b) \end{cases}$$

The f function is applied when a left child node is accessed: $\lambda_i^{left} = f(\lambda_i^{up}, \lambda_{i+2^d}^{up}), 0 \leq i < 2^d$. The g function is used when a right child node is accessed: $\lambda_i^{right} = g(\lambda_i^{up}, \lambda_{i+2^d}^{up}), 0 \leq i < 2^d$. Then moving up in the tree, the first half of partial sum is updated with $s_i^{up} = h(s_i^{left}, s_i^{right}), 0 \leq i < 2^d/2$ and the second half is simply copied : $s_i^{up} = s_i^{right}$. The decoding process stops when the partial sum of the root node is updated. In a systematic Polar encoding scheme, this partial sum is the decoded codeword. In practice, by exploiting knowledge on the frozen bits fixed location, whole sub-trees can be pruned and replaced by specialized nodes [14], [22], replacing scalar computations in the lowest levels of the tree by vector ones.

III. Software SC Decoders State-of-The-Art

In [14]–[16], SIMD units process several LLRs in parallel *within* a single frame decoding. This approach, called *intra-frame* vectorization is efficient in the upper layers of the tree and in the specialized nodes, but more limited in the lowest layers where the computation becomes more sequential.

In [17], [18], an alternative scheme called *inter-frame* vectorization decodes several independent frames in parallel in order to saturate the SIMD unit. This approach improves the throughput of the SC decoder but requires to load several frames before starting to decode, increasing both the decoding latency and the decoder memory footprint.

The AFF3CT software for SC decoding [19] is a multi-platform tool (x86-SSE, x86-AVX, ARM32-NEON, ARM64-NEON) including all state-of-the-art advances in software SC decoding of Polar codes: intra/inter-frame vectorization, multiple data formats (8-bit fixed-point, 32-bit floating-point) and all known tree pruning strategies. It resorts to code generation strategies to build specialized decoders, trading flexibility (code rate R, code length N) for extra performance.

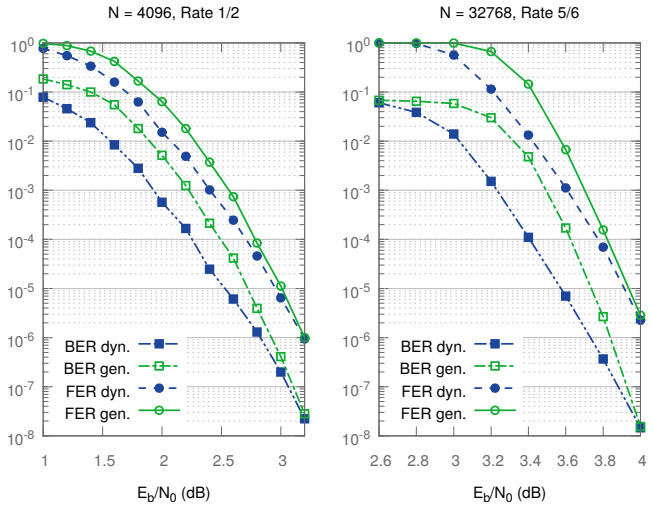

Fig. 2. Bit Error Rate (squares) and Frame Error Rate (circles) for the Fast-SSC decoder. The solid shapes represent the dynamic decoder with adaptive frozen bits when the hollow shapes represent the generated decoders. For $N = 4096$, the generated decoder is optimized for 3.2dB. For $N = 32768$, the generated decoder is optimized for 4.0dB.

All state of the art implementations aim at providing different trade-offs between error correction performance throughput and decoding latency. However, energy consumption is also a crucial parameter in SDR systems, as highlighted in [23]–[25]. In this study, we propose to investigate the influence of several parameters on the energy consumption of SC software Polar decoders on embedded processors to demonstrate their effectiveness for future SDR systems.

IV. Dynamic versus Generated Approach

We extend the AFF3CT software with a new version of the Fast-SSC decoder, called *dynamic* decoder. This version uses the same building blocks as the generated versions, but the same code is able to accommodate with different frozen bit layouts and different parameters (length, SNR). `C++11` template specialization features are used to enable the compiler to perform loop unrolling starting from a selected level in the decoding tree. It is the first non-generated version (to the best of our knowledge) to support both multi-precision (32-bit, 8-bit) and multi-SIMD strategies (intra-frame or inter-frame).

By design, generated decoders are still faster than the dynamic decoder (up to 20%). However each generated decoder is optimized for a single SNR. For very large frame sizes, the dynamic decoder outperforms generated decoders because the heavily unrolled generated decoders exceed Level 1 instruction cache size capacity [19].

Fig. 2 shows the Bit Error Rate (BER) and the Frame Error Rate (FER) of our dynamic and different generated decoders for $N = 4096$ and for $N = 32768$. Since there is almost no performance degradation between the 8-bit fixed-point decoders and the 32-bit floating-point ones, only 8-bit results are shown. We observe that the BER/FER performance

TABLE I
SPECIFICATIONS OF THE ODROID AND THE JUNO BOARDS.

	ODROID-XU+E	JUNO
SoC	Samsung Exynos 5410 (Exynos 5 Octa)	ARM64 big.LITTLE (dev. platform)
Arch.	32-bit, ARMv7	64-bit, ARMv8
Process	28nm	unspecified (32/28 nm)
big	4xCortex-A15 MPCore freq. [0.8-1.6GHz] L1I 32KB, L1D 32KB L2 2MB	2xCortex-A57 MPCore freq. [0.45-1.1GHz] L1I 48KB, L1D 32KB L2 2MB
LITTLE	4xCortex-A7 MPCore freq. [250-600MHz] L1I 32KB, L1D 32KB L2 512KB	4xCortex-A53 MPCore freq. [450-850MHz] L1I 32KB, L1D 32KB L2 1MB

is better for the dynamic version than for the generated codes. Indeed the generated versions are by definition optimized for a fixed set of frozen bits, and optimal for 3.2dB for $N = 4096$ and 4.0dB for $N = 32768$. As a result the generated versions are only competitive for a narrow SNR sweet spot. A decoder for a wider range of SNR values requires to combine many different generated versions.

V. EXPLORING PERFORMANCE TRADE-OFF

The objective and originality of this study is to explore different software and hardware parameters for the execution of a software SC decoder on modern ARM architectures. For a software decoder such as AFF3CT, many parameters can be explored, influencing performance and energy efficiency. The target rate and frame size are applicative parameters. The SIMDization strategies (intra-frame or inter-frame) and the features of decoders (generated or dynamic) are software parameters. The target architecture, its frequency and its voltage are hardware parameters. This study investigates the correlations between these parameters, in order to better choose the right implementation for a given applicative purpose. The low-power general purpose ARM32 and ARM64 processor test-beds based on big.LITTLE architecture are selected as representative of modern multi-core and heterogeneous architectures. The SC decoder is AFF3CT [19], enabling the comparison of different vectorization schemes.

The flexibility of the AFF3CT software allows to alter many parameters and turn many optimizations on or off, leading to a large amount of potential combinations. For the purpose of this study, computations are performed with 8-bit fixed-point data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in Joules to decode one bit of information, expressed as $E_b = (P \times l)/(K \times n_f)$ where P is the average power (Watts), l is the latency (s), K the number of information bits and n_f is the number of frames decoded in parallel (in the inter-frame implementation $n_f > 1$).

Testbed. The experiments are conducted on two ARM big.LITTLE platforms, an ODROID-XU+E board using a 32-bit Samsung Exynos 5410 CPU and the reference 64-bit JUNO Development Platform from ARM running a Linux operating system, detailed in Table I.

TABLE II
CHARACTERISTICS FOR EACH CLUSTER (T_i IS THE INFORMATION THROUGHPUT), FOR DYN. DECODER. $N = 4096$, RATE $R = 1/2$. THE RAM CONSUMPTION IS NOT INCLUDED IN E_b AND IN P.

Cluster	Impl.	T_i (Mb/s)	l (μs)	E_b (nJ)	P (W)
A7-450MHz	seq.	3.1	655	37.8	0.117
	intra	13.0	158	9.5	0.123
	inter	21.8	1506	6.0	0.131
A53-450MHz	seq.	2.1	966	29.0	0.062
	intra	10.1	203	7.0	0.070
	inter	17.2	1902	5.1	0.088
A15-1.1GHz	seq.	7.5	274	122.0	0.913
	intra	35.2	58	28.2	0.991
	inter	62.8	522	17.4	1.093
A57-1.1GHz	seq.	9.2	222	78.9	0.730
	intra	39.2	52	21.1	0.826
	inter	65.1	503	14.2	0.923
i7-3.3GHz	seq.	36.3	56.5	235.4	8.532
	intra	221.8	9.2	40.5	9.017
	inter	632.2	51.8	15.8	9.997

The big and the LITTLE clusters of cores on the ODROID board are on/off in a mutually exclusive way. The active cluster is selected through the Linux `cpufreq` mechanism. Both clusters can be activated together or separately on the JUNO board. Both platforms report details on supply voltage, current amperage, power consumption for each cluster. Only the ODROID platform reports details for the RAM. Consequently, most experiments have been primarily conducted on the ODROID platform to benefit from the additional insight provided by the RAM metrics.

VI. EXPERIMENTS AND MEASUREMENTS

Table II gives an overview of the decoder behavior on different clusters and for various implementations. The code is always single threaded and only the 8-bit fixed-point decoders are considered, since 32-bit floating-point versions are 4 times more energy consuming, on average. The sequential version is mentioned for reference only, as the throughput T_i is much higher on vectorized versions. Generally the inter-frame SIMD strategy delivers better performance at the cost of a higher latency l. Table II also compares the energy consumption of LITTLE and big clusters. The A53 consumes less energy than the A7 and the A57 consumes less energy than the A15, respectively. This can be explained by architectural improvements brought by the more recent ARM64 platform. Despite the fact that the ARM64 is a development board, the ARM64 outperforms the ARM32 architecture. Finally we observe that the power consumption is higher for the inter-frame version than for the intra-frame one because it fills the SIMD units more intensively, and the SIMD units consume more than the scalar pipeline.

For comparison, the results for the Intel Core i7-4850HQ, using SSE4.1 instructions (same vector length as ARM NEON vectors) are also included. Even if the i7 is competitive with the ARM big cores in term of *energy-per-bit* (Eb), these results show it is not well suited for the low power SDR systems because of its high power requirements. Table III shows a performance comparison (throughput, latency) with

TABLE III
COMPARISON OF 8-BIT FIXED-POINT DECODERS WITH INTRA-FRAME VECTORIZATION. $N = 32768$ AND $R = 5/6$.

Decoder	Platform	Freq.	SIMD	T_i (Mb/s)	l (μs)
[15]	i7-2600	3.4Ghz	SSE4.1	204	135
this work	i7-4850HQ	3.3Ghz	SSE4.1	**580**	47
this work	A15	1.1Ghz	NEON	70	391
this work	A57	1.1Ghz	NEON	73	374

the dynamic intra-frame decoder of [15]. On a x86 CPU, our dynamic decoder is 2.8 times faster than the state-of-the-art decoder. Even if we used a more recent CPU, we also used the same set of instructions (SSE4.1) and the frequencies are comparable.

Figure 3 shows the *energy-per-bit* consumption depending on the frame size N for the fixed rate $R = 1/2$. In general, the energy consumption increases with the frame size. For small frame sizes (N from 2^8 to 2^{14}), the inter-frame SIMD outperforms the intra-frame SIMD. This is especially true for $N = 2^8$ which has a low ratio of SIMD computations over scalar computations in the intra-frame version. As the frame size increases, the ratio of SIMD vs scalar computations increases as well. At some point around $N = 2^{16}$ the intra-frame implementation begins to outperform the inter-frame one, because the data for the intra-frame decoder still fits in the CPU cache, whereas the data of the inter-frame decoder does not fit the cache anymore. In our case (8-bit fixed point numbers and 128-bit vector registers) the inter-frame decoders require 16 times more memory than the intra-frame decoders. Then, for the frame size $N = 2^{20}$, both intra and inter-frame decoders now exceed the cache capacity and the RAM power consumption becomes more significant due to the increased number of cache misses causing RAM transactions. In general the code generation is effective on the intra-frame strategy whereas it is negligible on the inter-frame version of the code.

Considering those previous observations, it is more energy efficient to use inter-frame strategy for small frame sizes,

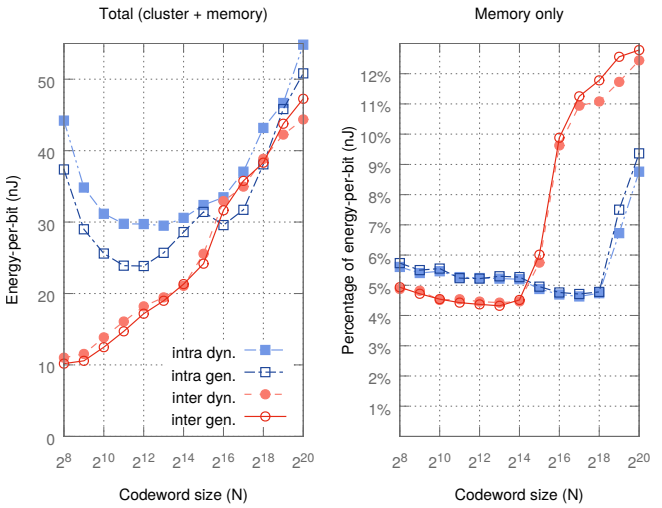

Fig. 3. Variation of the *energy-per-bit* for different frame sizes and impl.: intra-/inter-frame, dyn. code on/off, on A15 @ 1.1GHz. Fixed rate $R = 1/2$.

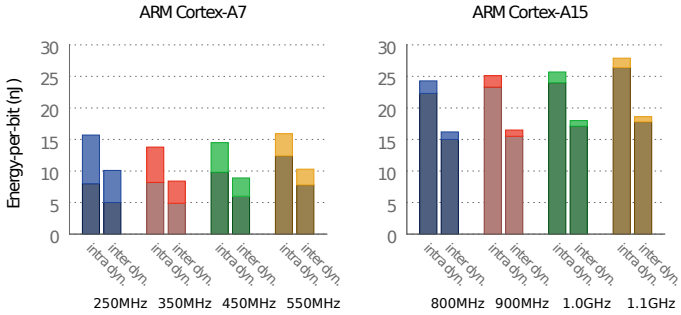

Fig. 4. Variation of the *energy-per-bit* (E_b) depending on the cluster frequency (dynamic code, intra-, inter-frame). A7 performance is on the left and A15 on the right. $N = 4096$ and $R = 1/2$. Dark colors and light colors stand for CPU cluster and RAM energy consumption, resp.

whereas it is better to apply intra-frame strategy for larger frame sizes (comparable energy consumption with much lower latency).

Figure 4 shows the impact of the frequency on the energy, for a given value of frame size $N = 4096$ and code rate $R = 1/2$. On both A7 and A15 clusters, the supply voltage increases with the frequency from 0.946V to 1.170V. The A7 LITTLE cluster shows that the energy consumed by the system RAM is significant: At 250MHz it accounts for half of the energy cost. Indeed, at low frequency, the long execution time due to the low throughput causes a high dynamic RAM refreshing bill. It is therefore more interesting to use frequencies higher than 250MHz. For this problem size and configuration, and from an energy-only point of view, the best choice is to run the decoder at 350MHz. On the A15 big cluster, the energy cost is mainly driven by the CPU frequency, while the RAM energy bill is limited compared to the CPU.

Thus, the bottom line about energy vs frequency relationship is: On the LITTLE cluster it is more interesting to clock the CPU at high frequency (higher throughput and smaller latency for a small additional energy cost); On the big cluster, where the RAM consumption is less significant, it is better to clock the CPU at a low frequency.

In Figure 5 the *energy-per-bit* cost decreases when the code rate increases. This is expected because there are many

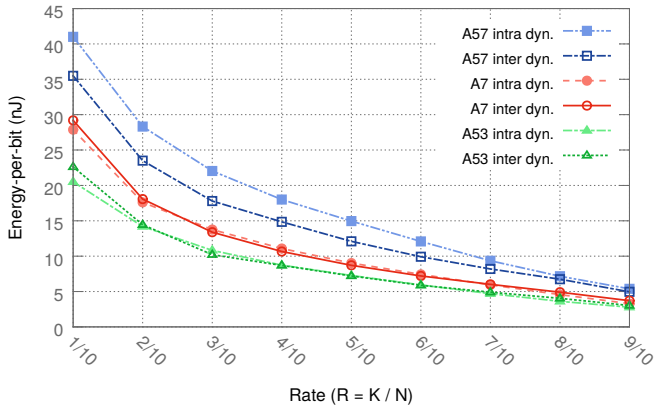

Fig. 5. Variation of the *energy-per-bit* (E_b) for $N = 32768$ depending on the rate $R = K/N$ (various impl.: intra-, inter-frame, code gen. on). Running on A7, A53 and A57 @ 450MHz.

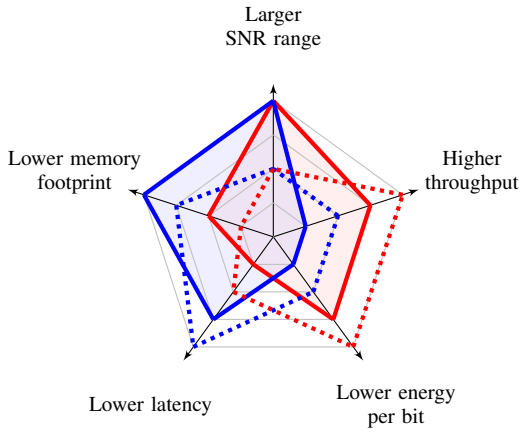

Fig. 6. Ranking of the different approaches along 5 metrics. In red, inter-frame vectorization performance and in blue, intra-frame performance. Solid color is for the dynamic versions, dotted is for the generated versions. Each version is sorted along each of the 5 axes and the best version for one axe is placed further from the center.

more information bits in the frame when R is high, making the decoder more energy efficient. With high rates, the SC decoding tree can be pruned more effectively, making the decoding process even more energy efficient. Figure 5 also compares the ARM A7, A53 and A57 clusters for the same 450MHz frequency (note: this frequency is not available on the A15). The LITTLE A7 is more energy efficient than the big A57, and the LITTLE A53 is itself more energy efficient than the LITTLE A7 ($E_{b_{A53}} < E_{b_{A7}} < E_{b_{A57}}$).

Figure 6 presents a qualitative summary of the characteristics of the different code versions, for intra-/inter-frame vectorization, generated or dynamic code. For instance, if the size of the memory footprint is an essential criterion, the dynamic intra-frame code exhibits the best performance.

To sum up, the dynamic implementations provides efficient trade-off between throughput, latency and energy depending on code length. It was demonstrated by previous benchmarks. Both implementations provide low-energy and low-power characteristics compared to previous works in the field on x86 processors [14]–[19]. Whereas the throughput on a single processor core is reduced compared to x86 implementations, ARM implementations must fulfil a large set of SDR applications with limited throughputs and where the power consumption matters. Finally, it is important to notice that multi-core implementations of the proposed ARM decoders is still possible on these ARM targets to improve the decoding throughputs.

VII. Conclusion and Future Work

This paper presented for the first time a study comparing performance and energy consumption for software Successive Cancellation Polar decoders on big.LITTLE ARM32 and ARM64 processors. We proposed a new decoder implementation, and showed how decoding performance, throughput and decoder implementation correlate for a range of applicative parameters, software optimizations and hardware architectures.

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).

References

[1] G. Wang, M. Wu, B. Yin, and J. R. Cavallaro, "High throughput low latency LDPC decoding on GPU for SDR systems," in *Proc. of the IEEE GlobalSIP Conf.*, 2013.

[2] B. Le Gal and C. Jego, "High-throughput multi-core LDPC decoders based on x86 processor," *IEEE TPDS*, vol. 27, no. 5, pp. 1373–1386, 2016.

[3] B. L. Gal and C. Jego, "High-throughput LDPC decoder on low-power embedded processors," *IEEE Comm. Letters*, vol. 19, no. 11, pp. 1861–1864, 2015.

[4] D. Yoge and N. Chandrachoodan, "GPU implementation of a programmable turbo decoder for software defined radio applications," in *Proc. of the IEEE VLSI Design Conf.*, 2012.

[5] M. Wu, G. Wang, B. Yin, C. Studer, and J. Cavallaro, "HSPA+/LTE-A turbo decoder on GPU and multicore CPU," in *Proc. of the IEEE ACSSC*, 2013.

[6] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE TIT*, vol. 55, no. 7, pp. 3051–3073, 2009.

[7] A. Mishra, A. J. Raymond, L. G. Amaru, G. Sarkis, C. Leroux, P. Meinerzhagen, A. Burg, and W. Gross, "A successive cancellation decoder ASIC for a 1024-bit polar code in 180nm CMOS," in *Proc. of the IEEE A-SSCC*, 2012.

[8] C. Leroux, A. J. Raymond, G. Sarkis, I. Tal, A. Vardy, and W. J. Gross, "Hardware implementation of successive-cancellation decoders for polar codes," *Springer JSPS*, vol. 69, no. 3, pp. 305–315, 2012.

[9] A. Raymond and W. Gross, "A scalable successive-cancellation decoder for polar codes," *IEEE TSP*, vol. 62, no. 20, pp. 5339–5347, 2014.

[10] B. Li, H. Shen, D. Tse, and W. Tong, "Low-latency polar codes via hybrid decoding," in *Proc. of the IEEE ISTC Symp.*, 2014.

[11] B. Yuan and K. Parhi, "Low-latency successive-cancellation polar decoder architectures using 2-bit decoding," *IEEE TCS*, vol. 61, no. 4, pp. 1241–1254, 2014.

[12] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE JSAC*, vol. 32, no. 5, pp. 946–957, 2014.

[13] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "A 237 Gbps unrolled hardware polar decoder," *Electronics Letters*, vol. 51, no. 10, pp. 762–763, 2015.

[14] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE JSAC*, vol. 32, no. 5, pp. 946–957, 2014.

[15] P. Giard, G. Sarkis, C. Thibeault, and W. Gross, "Fast software polar decoders," in *Proc. of the IEEE ICASSP*, 2014.

[16] G. Sarkis, P. Giard, C. Thibeault, and W. Gross, "Autogenerating software polar decoders," in *Proc. of the IEEE GlobalSIP Conf.*, 2014.

[17] B. Le Gal, C. Leroux, and C. Jego, "Software polar decoder on an embedded processor," in *Proc. of the IEEE SiPS Work.*, 2014.

[18] ——, "Multi-Gb/s software decoding of polar codes," *IEEE TSP*, vol. 63, no. 2, pp. 349–359, 2015.

[19] A. Cassagne, B. Le Gal, C. Leroux, O. Aumage, and D. Barthou, "An efficient, portable and generic library for successive cancellation decoding of polar codes," in *Proc. of the Springer LCPC Work.*, 2015.

[20] AFF3CT, "AFF3CT: The first software release," 2016. [Online]. Available: http://dx.doi.org/10.5281/zenodo.55668

[21] E. Arikan, "Systematic polar coding," *IEEE Comm. Letters*, vol. 15, no. 8, pp. 860–862, 2011.

[22] A. Alamdar-Yazdi and F. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Comm. Letters*, vol. 15, no. 12, pp. 1378–1380, 2011.

[23] A. M. Wyglinski, M. Nekovee, and T. Hou, *Cognitive radio communications and networks: principles and practice*. Academic Press, 2009.

[24] P. Dutta, Y.-S. Kuo, A. Ledeczi, T. Schmid, and P. Volgyesi, "Putting the software radio on a low-calorie diet," in *Proc. of the ACM SIGCOMM HotNets Work.*, 2010.

[25] S. Shaik and S. Angadi, "Architecture and component selection for SDR applications," *IJETT*, vol. 4, no. 4, pp. 691–694, 2013.