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:

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 (multiprecision 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† [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.

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], Arikan 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: \(U = V \ast \kappa \otimes^n\) and bits of \(U\) in the frozen location are replaced by zeros. The codeword is finally obtained with a second encoding phase: \(X = U \ast \kappa \otimes^n\).

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

$$\lambda^0_i = \log \frac{\Pr(Y, \hat{u}_{i+1}|u_i = 0)}{\Pr(Y, \hat{u}_{i+1}|u_i = 1)}.$$ 

The estimated bit $\hat{u}_i$ is 0 if $\lambda^0_i > 0$, 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 $\lambda^0_i$ 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 $s_i$. Nodes are visited using a pre-order traversal. As shown in Figure 1, three functions, $f$, $g$ and $h$ are used for node updates:

$$f(\lambda_a, \lambda_b) = \text{sign}(\lambda_a, \lambda_b), \min(|\lambda_a|, |\lambda_b|)$$

$$g(\lambda_a, \lambda_b, s) = (1 - 2s)\lambda_a + \lambda_b$$

$$h(s_a, s_b) = (s_a \oplus s_b, s_b)$$

The $f$ function is applied when a left child node is accessed: $\lambda_i^{left} = f(\lambda_{i+2d}, S_{i+2d}), 0 \leq i < 2^d$. The $g$ function is applied when a right child node is accessed: $\lambda_i^{right} = g(\lambda_{i+2d}, S_{i+2d}), 0 \leq i < 2^d$. Then moving up in the tree, the first half of partial sum is updated with $s_{i+2d} = h(s_i^{left}, s_i^{right}), 0 \leq i < 2^d/2$ 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 layers 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.

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.

<table>
<thead>
<tr>
<th>SoC</th>
<th>Arch.</th>
<th>Process</th>
<th>ODOROID-XU+E</th>
<th>JUNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM64 big.LITTLE</td>
<td>32-bit, ARMv7</td>
<td>28nm (unspecified)</td>
<td>L1I 32KB, L1D 32KB</td>
<td>L1I 32KB, L1D 32KB</td>
</tr>
<tr>
<td>freq. [0.8-1.6GHz]</td>
<td>freq. [250-600MHz]</td>
<td>freq. [250-600MHz]</td>
<td>L2 2MB</td>
<td>L2 1MB</td>
</tr>
<tr>
<td>freq. [250-600MHz]</td>
<td>freq. [450-850MHz]</td>
<td>freq. [450-850MHz]</td>
<td>L1I 32KB, L1D 32KB</td>
<td>L1I 32KB, L1D 32KB</td>
</tr>
<tr>
<td>freq. [250-600MHz]</td>
<td>freq. [450-850MHz]</td>
<td>freq. [450-850MHz]</td>
<td>L2 512KB</td>
<td>L2 1MB</td>
</tr>
</tbody>
</table>

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 data types, with all tree pruning optimizations activated. 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 data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric considered is the average amount of energy in data types, with all tree pruning optimizations activated. The main metric consid...
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
$N = 2^{16}$ 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,
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

---

### Table III

<table>
<thead>
<tr>
<th>Decoder</th>
<th>Platform</th>
<th>Freq.</th>
<th>SIMD</th>
<th>$T_1$ (Mb/s)</th>
<th>$l$ (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[15]</td>
<td>i7-2600</td>
<td>3.4GHz</td>
<td>SSE4.1</td>
<td>204</td>
<td>135</td>
</tr>
<tr>
<td>this work</td>
<td>i7-4850HQ</td>
<td>2.3GHz</td>
<td>SSE4.1</td>
<td>580</td>
<td>70</td>
</tr>
<tr>
<td>this work</td>
<td>A57</td>
<td>1.1GHz</td>
<td>NEON</td>
<td>73</td>
<td>374</td>
</tr>
</tbody>
</table>

---

**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 $R = 1/2$.
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 ($E_{b_{A53}} < E_{b_{A57}} < E_{b_{A7}}$).

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 fulfill 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