Approximated Canonical Signed Digit for Error Resilient Intelligent Computation

Cardarilli, Gian Carlo; Di Nunzio, Luca; Fazzolari, Rocco; Nannarelli, Alberto; Re, M.

Published in:
Proceedings of 53rd Asilomar Conference on Signals, Systems, and Computers

Link to article, DOI:
10.1109/IEEECONF44664.2019.9049051

Publication date:
2020

Document Version
Peer reviewed version

Citation (APA):
Approximated Canonical Signed Digit for Error Resilient Intelligent Computation

G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, A. Nannarelli(1) and M. Re

Department of Electronics, University of Rome Tor Vergata, Rome, Italy
(1)DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark

Abstract—Lowering the energy consumption in applications operating on large datasets is one of the main challenges in modern computing. In this context, it is especially important to lower the energy required to transfer data from/to the memory. Usually, this is obtained by applying smart encoding techniques to the data. In this work, we show how to reduce the switching activity in buses and floating-point units by an approximated canonical signed-digit encoder. The precision of the encoding is programmable and can be chosen depending on the application’s required accuracy.

I. INTRODUCTION

Lowering the energy consumption in architectures implementing Intelligent Computation algorithms is a primary target for new applications such as, for example, Internet-of-Things (IoT). This goal is achieved by using architectural and micro-electronic technology level techniques. At architectural level, the most important actions are aimed at minimizing power dissipation in parts characterized by high capacitive loads due to interconnect (buses), or due to complex architectures, such as hardware multipliers.

Data on power dissipation break-down in multicore chips show that the cores consume about 50% of the total power, and that the several levels of memories account for the remaining 50% [1].

In [2], the authors show that for a 45nm CMOS process the memory access measured in [pJ] is three order of magnitude more expensive than the arithmetic implemented in the core. Consequently, energy reduction in memory access is very important for lowering power consumption in applications with large datasets, e.g., artificial neural networks (ANNs).

In [3] and [4], the authors show different techniques for the implementation of low power interconnects by using Canonical Signed Digit Number System and the Binary Coded Canonical Digit Signed Digit. In [5], the authors use Canonical Signed Digit (CSD) coding to reduce the time of execution of the multiplications that are massively used in the ANN inference phase, while in [6], an ANN architecture is implemented on FPGA using CSD-based multipliers obtaining a reduction in hardware resources.

Moreover, a number of engineering fields deal with error resilient applications (image and audio processing) and algorithms (ANN, fuzzy logic, genetic algorithms). Consequently, techniques such as approximate arithmetic and approximate data representation can be fruitfully applied in these areas to achieve energy savings without sacrificing algorithm performances.

As examples, an energy efficient neuromorphic architecture based on approximated multipliers, and a systematic design space exploration approach based on approximated ANNs are presented in [7] and [8].

To address energy efficiency in computation for ANNs, we introduced in [9] a floating-point variable precision format, the Tunable Floating-Point (TFP) format, to handle several precisions and dynamic ranges. Moreover, TFP units for addition and multiplication were introduced in [10] and [11].

In this work, we propose a method to reduce the switching activity in the processing units and in the memory buses by an approximate CSD encoding.

The idea is to suppress sequences of ‘1’s by CSD recoding and approximate the encoded number by omitting the bits of negative weight. Fig. 1 shows an example of the proposed Approximate CSD (ACSD) recoding. Sequences of three or two ‘1’ are not recoded.

The main contribution of this work is the design of the ACSD coder, including a parallel CSD recoding algorithm. The coder precision can be programmed to adjust the accuracy of the approximated recoding for different applications during the algorithm execution. The ACSD coder is implemented in standard cells technology, and we provide data on performance, error rates, and power savings for a few test cases.

Simulations of traffic on the bus for several sample sequences of binary32 (single-precision) vectors with ACSD encoded significand show a best-case reduction in switching activity of 6%, and a worst-case reduction of 2%.

Similarly, the execution of matrix multiplication for ACSD encoded operands in a binary32 floating-point unit results in average power savings of about 3%.

II. PARALLEL RECODING ALGORITHM

Canonical recoding of binary numbers into the digit set \{-1, 0, 1\} is a sequential recoding that minimizes the number of non-zero bits [12]. For example, \(A = 01 \ 0111 \ 1001\) is recoded into \(F = 01 \ 1000 \ 0001\), where \(T = -1\).

In our algorithm, to parallelize the recoding, the binary number is split into 4-bit digits as illustrated in Fig. 2. To simplify the recoding, we recode only sequences with at least four ‘1’s. Since a sequence of ‘1’s can spread across more digits, we use the transfer bits \(Q\) and \(T\) to indicate:
Moreover, if there is a sequence of ‘1’s active in digit 4-bit digit 4-bit digit
length \( L_i \) We do not use indices to refer to local variables in digit 4-bit digit
describe how the recoding is done for the 4-bit digit 4-bit digit 4-bit digit.

Moreover, if there is a sequence of ‘1’s in digit 4-bit digit 4-bit digit, its length 4-bit digit 4-bit digit 4-bit digit is passed to digit 4-bit digit 4-bit digit 4-bit digit. Based on these definitions, we describe how the recoding is done for the 4-bit digit 4-bit digit 4-bit digit (Fig. 3). We do not use indices to refer to local variables in digit 4-bit digit 4-bit digit.

The recoding algorithm is divided in the following logical steps.

**Step 1.** The sequence of ‘1’s can be either toward the right or the left, or toward both ends. Therefore, we list in Table I the sequence lengths \( L \) (left) toward digit 4-bit digit 4-bit digit 4-bit digit +1 and \( R \) (right) toward digit 4-bit digit 4-bit digit 4-bit digit.

For example, when \( A=1101 \) there is a sequence of two ‘1’s toward the digit at left (\( L = 2 \)) and a sequence of one toward the digit at right (\( R = 1 \))

The sequence toward the left is passed to digit 4-bit digit 4-bit digit 4-bit digit

\[ L_i \leftarrow L = (L1L0)_2 \quad \text{(two bits)} \]

Moreover, the bit \( Q_i \) is set to one if \( A=1111 \).

**Step 2.** The results of Table I and the incoming \( Q_{i-1} \) and \( L_{i-1} \) are used to compute the length of the sequence in digit 4-bit digit 4-bit digit, as follows. We compute the length \( p \) of the sequence of ‘1’s

\[ p = p_{\text{local}} + p_R \]

where

- \( p_{\text{local}} \) is the length in the digit, computed as:

\[ p_{\text{local}} = 4 \quad \text{if } Q_i = 1 \]
\[ p_{\text{local}} = R \quad \text{otherwise} \]

In the latter case (\( Q_i = 0 \)), there is some ‘0’ in the digit and there is a sequence \( \geq 4 \) only if started in digit 4-bit digit 4-bit digit.

**Step 3.** Bit \( T_i \) and the incoming (from left) \( T_{i+1} \) determine the recoding in the digit. The actual recoding, i.e., transforming the sequence of ‘1’, is done according to \( L \) and \( R \) values determined in Table I.

![Fig. 2. Architecture of 4-bit digit-based recoder.](image)

![Fig. 3. Detail of the basic block and transfer bits.](image)

**TABLE I**

<table>
<thead>
<tr>
<th>A</th>
<th>L1</th>
<th>L0</th>
<th>R1</th>
<th>R0</th>
<th>Q_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0111</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1001</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1010</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1101</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1110</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**SEQUENCE LENGTHS:** \( \left( \begin{array}{l} L1L0 \end{array} \right)_2 \) holds the length toward the digit at left, \( \left( \begin{array}{l} R1R0 \end{array} \right)_2 \) toward the digit at right.
Extending the previous example, \( A = 1001\ 1110 \), we have:
\[
L_{i-1} = 3, \quad p = 4, \quad T_i = 1 \text{ resulting in } F = 1010\ 00\!00.\]
Clearly, we need a special code to represent the position of the approximated recoder.

The transfer bits \( Q_i \) and \( T_i \), and the \( L_{i-1} \) value do not propagate further than one digit, therefore, the recoding is done in parallel. The detail is illustrated in Fig. 3.

The recoder of Fig. 2 requires additional output bits to represent the bits with negative weights. These outputs are enabled by an enable signal (EN) to enforce the NRZ. The operations in the decoder are the following:

- In each digit of the recoder, we need to introduce an enable signal (EN) to enforce the NRZ. The approximation introduces an error \( 2^j \) where \( j \) is the position of \( T \) in the word (we assume unsigned integers, in this case).

**A. Non-Recoding Zone**

Since the error can be quite large if the omitted \( T \) is in the most-significant part of the word, we can adjust the accuracy of the approximated recoding by introducing a “non-recoding zone (NRZ)”.

Fig. 4 shows an example where the NRZ is extended to the 16 most-significant bits (MSBs). With this protection, \( H = 16 \), the maximum error is \( 2^7 \) (unsigned integers). The actual error in the example of Fig. 4 is \( 2^6 \).

The NRZ is specified by a mask \( M \), implemented by a decoder. For example, for a 16-bit integer, if we want to limit the error to \( 2^8 \) we have:

\[
A = 0011\ 1110\ 0111\ 1110 \quad \text{(input)}
\]
\[
M = 1111\ 1111\ 0000\ 0000 \quad \text{(mask)}
\]
\[
F = 0011\ 1110\ 1000\ 00\!00 \quad \text{(recoded)}
\]
\[
F_A = 0011\ 1110\ 1000\ 0000 \quad \text{(approx. recoded)}.
\]

In each digit of the recoder, we need to introduce an enable signal (EN) to enforce the NRZ. The operations in the modified 4-bit digit encoder, illustrated in Fig. 5, are the following:

- In each digit, the mask is checked against the position of the trailing ‘1’ (to become \( T \) and, therefore, omitted). For example, if \( A = 1110 \) we define the position of the trailing ‘1’ as \( B = 0010 \) and we determine if the position falls in the NRZ for each of the four bits \( B_j \) and \( M_j \)

\[
K = \neg(B_3M_3 \text{ or } B_2M_2 \text{ or } B_1M_1 \text{ or } B_0M_0)
\]

Therefore, \( K = 0 \) indicates that there is potential trailing ‘1’ falling in the NRZ. For example, for \( B = 0010 \), if \( M = 1111 \rightarrow K = 0 \), but if \( M = 1100 \rightarrow K = 1 \), and the potential trailing ‘1’s can be recoded because the error introduced by omitting \( T \) falls outside the NRZ.

- If \( K = 1 \), the partial sequence ‘1’ is recoded only if \( T_{i+1} = 1 \). In this case the “enable recoding” information is communicated to the adjacent digits if \( E_i = T_{i+1} \text{ AND } K \).
- To preserve the NRZ across several digits, long sequences of ‘1’, the enable in digit \( i + 1 \) is generated as:

\[
EN_{i+1} = E_i \text{ AND } E_{i-1} \text{ AND } \ldots \text{ AND } E_0
\]

**B. Critical Path**

Fig. 6 shows the architecture for a (n+1)-bit ACSD recoder with NRZ. The highlighted blocks contributes to the critical path. In two adjacent digits \( i \) and \( i + 1 \), there is competition between the sum of delays in **Step 1** and **Step 2** in block \( i + 1 \) (producing \( T_{i+1} \), and the delay of \( K \comp \)), in block \( i \) plus the delay of the decoder to generate the mask \( M \) depending on the value of the non-recoding zone \( H \), not depicted in the figure. Since the synthesis tends to equalize the slacks in the different paths, we reported both possibilities in Fig. 6.

Once all \( E_i \) values are generated in parallel (same delay for all digits), the longest delay in the critical path depends on the fan-in of the AND gate generating the \( EN_i \) values. Therefore, \( EN_n \) in the most-significant digit is the one on the critical path because the fan-in of the AND gate is \( n \) (the largest).

Fig. 6 demonstrates that in the overall recoding process with non-recoding zone there is no carry propagation through the \( n+1 \) digits.

**III. Experimental Results**

We ran some tests to evaluate the errors introduced by the approximated recoder for approximating the 24-bit significand of binary32 (single-precision) floating-point numbers. This format is suitable for training in deep learning applications.

We ran tests on 1024 24-bit random vectors and extended the “non-recoding zone” from \( H = 0 \) (fully approximated) to \( H = 24 \) (no recoding).
The results, reported in Table II, show that the average error on the significand $\epsilon_{ave}$ is quite acceptable and the reduction in the number of ‘1’s in the representation is quite significant. Moreover, for each NRZ, the table reports the number of approximated elements (col. 2), and the reductions in the number of ‘1’s for the 1024 elements after recoding (col. 5).

The reduction in the switching activity in a bus depends on how the sequence of data are accessed (transit on bus). We ran simulations for several sample sequences (traffic on the bus) and found a best-case reduction in switching activity of 6%, and a worst-case reduction of 2%. Although the reduction is small, it may lead to sizeable power dissipation savings in the bus if the switched capacitance is large.

A. Hardware Implementation

The 24-bit approximated recoder is implemented in a commercial 45 nm library of standard cell. The unit includes hardware to renormalize the significand in case the recoding causes an overflow.

The latency of the approximated recoding is 520 ps, corresponding to 8 FO4 delay.

As application, we chose matrix multiplication, arguably, the most common kernel in machine learning.

Fig. 7 shows the average power dissipation reduction when the approximated recoding is applied to a FP-unit (FP-multiplier and FP-adder) executing the matrix product. By increasing the approximation (reducing $H$) we can achieve power savings of about 3%.

IV. Conclusions

In this work, we designed a parallel approximated CSD recoder to reduce the switching activity in buses and FP-units.

The recoding is done in parallel on 4-bit digits and the overall operation is carry-free. The critical path traverses the blocks belonging to three digits, but contributing to the delay of one digit only, plus a few gates (Fig. 6).

Although the energy savings are not large, the trade-off error/savings makes the ACSD recoder suitable for applications in deep learning and other areas where huge datasets need to be transferred from/to memory.
REFERENCES


