Nonconventional Computer Arithmetic
Circuits, Systems and Applications
Leonel Sousa, Senior Member, IEEE

Abstract—Arithmetic plays a major role in a computer’s performance and efficiency. Building new computing platforms supported by the traditional binary arithmetic and silicon-based technologies to meet the requirements of today’s applications is becoming increasingly more challenging, regardless whether we consider embedded devices or high-performance computers. As a result, a significant amount of research effort has been devoted to the study of nonconventional number systems to investigate more efficient arithmetic circuits and improved computer technologies to facilitate the development of computational units that can meet the requirements of applications in emergent domains. This paper presents an overview of the state of the art in nonconventional computer arithmetic. Several different alternative computing models and emerging technologies are analyzed, such as nanotechnologies, superconductor devices, and biological- and quantum-based computing, and their applications to multiple domains are discussed. A comprehensive approach is followed in a survey of the logarithmic and residue number systems, the hyperdimensional and stochastic computation models, and the arithmetic for quantum- and DNA-based computing systems and techniques for approximate computing. Technologies, processors and systems addressing these nonconventional computer arithmetic systems are also reviewed, taking into consideration some of the most prominent applications, such as deep learning or postquantum cryptography. In the end, some conclusions are drawn, and directions for future research on nonconventional computer arithmetic are discussed.

I. INTRODUCTION

The demise of Moore’s Law and the waning of Dennard scaling, which respectively stated that the number of transistors on silicon chips would double every two years and that this increase in the transistor density is not achieved at constant power consumption [1], mark the end of an era in which the computational capacity growth was mainly based on the downscaling of silicon-based technology. At the same time, the demand for data processing is higher than ever before as a result of the more than 2.5 quintillion bytes that are created on a daily basis [2]. This number continues to grow exponentially, and therefore, ingenious solutions must be developed to address the limitations of traditional computational systems. These innovative solutions must include developments not only at the technological level but also at the arithmetic, architectural and algorithmic levels of computing.

Although binary arithmetic has been successfully used to design silicon-based computing systems, the positional nature of this representation imposes the processing of carry chains, which precludes the exploitation of parallelism at the most basic levels and leads to high power consumption. Hence, the research on unconventional number systems is of the utmost interest to explore parallelism and take advantage of the characteristics of emerging technologies to improve both the performance and the energy efficiency of computational systems. Moreover, by avoiding the dependencies of binary systems, nonconventional number systems can also support the design of reliable computing systems using the newest available technologies, such as nanotechnologies.

A. Motivation

The Complementary Metal-Oxide Semiconductor (CMOS) transistor was invented over fifty years ago and has played a key role in the development of modern electronic devices and all that it has enabled. The CMOS transistor has evolved into nanodevices, with characteristic dimensions less than 100 nm. The downscaling of the gate length has become one of the biggest challenges hindering progression in each new generation of CMOS transistors and integrated circuits. New device architectures and materials have been proposed to address this challenge, namely, the Fin Field-Effect transistor (FinFET) multigate devices [5]. This type of nonplanar transistor became the dominant gate design from the 14 nm/10 nm generations, used in a broad range of applications, ranging from consumer applications to embedded systems and high-performance computing [6].

Fig. 1: Emerging technologies and CMOS speed, size, cost, and switching energy (scales are logarithmic) [7]

Fig. 1 plots the cost, speed, size, and energy per operation relationship for the CMOS and other emerging nanotechnologies; all the scales are logarithmic, covering many orders of magnitude. Examples of these technologies include superconducting electronics, molecular electronics, resonant tunneling devices, quantum cellular automata, and optical switches.
Superconducting digital logic circuits use Single-Flux Quantum (SFQ), also known as magnetic flux quanta, to encode and process data. An Rapid Single Flux Quantum (RSFQ) device is a superconducting ring with a Josephson junction that opens and closes to admit or expel a flux quanta. The Adiabatic Quantum-Flux-Parametron (AQFP), another SFQ-based device not represented in the figure with characteristics that are analyzed later in this survey, drastically reduces the energy dissipation by using Alternating Current (AC) bias/excitation currents for both the clock signal and power supply. This is high-speed nanotechnology that offers low-power dissipation and scalability. However, its main drawbacks include the need for expensive cooling technologies and improved techniques in manufacturing the elements. Molecular electronics use a single molecule as the switch. The configuration of the molecule determines its state, thereby “switching” on or off the current flow through the molecule. While conceptually appealing, the molecular process exhibits poor self-assembly techniques and low reproducibility. In Quantum Cellular Automata (QCA), cells are switched on and off by moving the position of the electrons from one cell to another. Classical physics does not apply to these technologies, which support, for example, quantum computing. While it is challenging to build general purpose computing engines based on these technologies, they are much faster on certain specific tasks. A last example is the integrated optical technology, which is based on having small optical cavities that change their properties to either store or emit photons – a zero or a one. It is a challenging approach, both from the fan-out and cost point of view. As will be seen, computing-in-the-network circuits can be designed with integrated photonics switches.

SECTION II: COMPUTER ARITHMETIC AND ARCHITECTURES

These emergent technologies will become more and more relevant as the end of the CMOS era approaches [8]. Nevertheless, for the time being, there is no technology better than CMOS, as it provides a good balance between energy consumption, cost, speed and scalability, as shown in Fig. 1. Yet, better solutions can be identified in Fig. 1 along every individual axis. However, to design and implement processing systems based on these new technologies and devices, different solutions have to be found at various levels: from the circuits to the data representation and architectures, as depicted in Fig. 2. For the conventional scaled CMOS technology, the layers have been quite well established, and the characteristics were successfully explored from the material layer, silicon, up to the topmost layer, multicore architectures. Moreover, the conventional binary system has been dominant at the data representation layer. However, when considering emergent technologies or new architectures for designing these systems, data representation must also be reconsidered to properly adapt each system to the features of both the bottom and top layers. Data representation and the related computer arithmetic are the motivation behind this survey, which looks beyond the CMOS technology and conventional systems.

B. Structure of the survey

This survey analyzes the confluence of emergent technologies, nonconventional computer arithmetic, innovative computing paradigms and new applications. Similar to a jigsaw puzzle, although these pieces exist by themselves, it is only by connecting them in the right way that the whole picture can be derived and, in this particular case, the way to design efficient systems can be determined. Fig. 3 depicts the tight interconnections and dependencies of all these aspects in a matrix representation. In addition, this figure shows how these very important topics are addressed in this paper, highlighting the fact that nonconventional arithmetic exposes the characteristics of the underlying technology to assist in the development of efficient and reliable computing systems useful for different applications.

Fig. 3: Structure of the survey: sections and topics

Section 1 introduces nonconventional number systems, the main characteristics and the associated computer arithmetic. The Logarithmic Number System (LNS) and the Residue Number System (RNS) shorten up the carry chain underlying binary arithmetic. While LNS compresses the number representation significantly, mitigating the problem of carry chains, it lowers the complexity of the multiplication, division, and square-root arithmetic operations [11]. RNS [4] is a type of nonpositional number system in which integers are represented as a set of smaller residues that can be processed independently and in parallel. Stochastic Computing (SC) and Hyper-Dimensional Computing (HDC) introduce robustness against randomness in number representation. SC represents the probability of a bit taking the value ‘1’ or ‘0’ in a bitstream, regardless of its position [56], processing data bitwise with simple circuits. HDC operates on very large binary vectors.
assuming that information is represented in a highly structured way using hyperdimensional vectors, as in the human brain, where details are not very important for data processing [70].

Section II is devoted to the analyses of new technologies and computing paradigms. Nanotechnologies, such as spin transistors, superconducting electronics, molecular electronics, and resonant tunneling devices, have emerged as alternative technologies to CMOS [1], while Deoxyribonucleic Acid (DNA)-based computing [10] and Quantum Computing (QC) [155] are new computing paradigms. It is shown that nonconventional arithmetic is fundamental not only in the design of efficient computer systems based on these new paradigms and technologies but also in the mitigation of some of its intrinsic disadvantages. For example, it is shown that the RNS is useful in designing energy-efficient integrated photonics-based computational units and in overcoming the negative effects caused by the instability of the biochemical reactions and the error hybridizations in DNA computing [113], as highlighted in Fig. 3. Furthermore, it is discussed how SC can be applied to mitigate the deep pipelining nature of the AQFP logic devices, while HDC may support computing nanosystems through the heterogeneous integration of multiple emerging nanotechnologies [106].

In Section IV two classes of applications are considered as examples of use cases that highly benefit from nonconventional arithmetic: machine learning and postquantum cryptography. While quantum-resistant cryptography makes use of RNS and SC, as shown in Fig. 3, machine learning applications may take advantage of all the classes of nonconventional arithmetic considered in this paper. For example, the Deep Learning (DL) framework that adopts AQPFP to achieve high energy efficiency with superconductivity is an example of SC to machine learning on emerging technologies [79].

The goal of this paper is to provide a survey on nonconventional computer arithmetic circuits and systems, describing their main characteristics, mathematical tools and algorithms, and to discuss architectures and technologies for their implementation. We follow a tutorial approach on how to use nonconventional arithmetic to compute emergent applications and systems, and in the end, we draw some conclusions and ideas for future research work. We find that this work can be useful for engineers and researchers in computational arithmetic, in particular as a source of inspiration for doctoral students. We also provide more than one hundred and seventy references to the most important works related to nonconventional computer arithmetic, distributed according to the main topics and sections presented in Fig. 3.

II. NONCONVENTIONAL NUMBER SYSTEMS: ARITHMETIC AND ARCHITECTURES

This section introduces the LNS, RNS, SC and HDC nonconventional representations and the associated arithmetic properties. These properties are then leveraged in the design of arithmetic units found in many processors.

A. Logarithmic number systems

The LNS has been proposed for both the fixed-point [10] and floating-point [11] formats. In the standardized IEEE 754 single-precision Floating Point (FP) format, a number $P$ may assume the normalized absolute values in Table I. For a 24-bit mantissa, if the hidden bit is counted, the effective resolution is between one part in $2^{24}$ and one part in $2^{23}$, approximately $10^{-7}$. Note that the boundary values for the exponent (0 and 255) are reserved for special cases.

<table>
<thead>
<tr>
<th>$P_{FP}$ (sign bit)</th>
<th>$E_{FP}$ (exponent): 8 bits</th>
<th>$F_{FP}$ (mantissa): 23 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{LNS}$ (sign bit)</td>
<td>$T_{LNS}$ (integer): 8 bits</td>
<td>$F_{LNS}$ (fractional): 23 bits</td>
</tr>
<tr>
<td>$P = -1^{F_{FP}} \times 1.F_{FP} \times 2^{E_{FP}-127}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p = -1^{F_{LNS}} \times 2^{T_{LNS}} \times F_{LNS}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Absolute values</th>
<th>minimum</th>
<th>maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$ (FP)</td>
<td>$2^{-128} \approx 1.2 \times 10^{-38}$</td>
<td>$2^{127} \approx 3.4 \times 10^{38}$</td>
</tr>
<tr>
<td>$p$ (LNS)</td>
<td>$2^{-128} \approx 2.9 \times 10^{-39}$</td>
<td>$2^{128} \approx 3.4 \times 10^{38}$</td>
</tr>
</tbody>
</table>

FP is a “semi-logarithmic” representation with a fixed-point component – significand or mantissa – that is scaled by a factor $P$ – the exponent – that is represented by its logarithmic value in base 2. In comparison, the same number $P$ is represented in LNS by the 3-tuple $(b, s, p)$, where $b$ is the adopted logarithm base (without lack of generality, we consider the typical case $b = 2$), the $s$ bit denotes the sign, and $p = \log_2 |P|$. The exponent can be computed from the integer and fractional parts. This fixed-point format provides a range of numbers similar to the FP: this range is defined by the $IT$ bits, while the width $F$ controls the precision. The logarithmic representation may be beneficial to applications with data and/or parameters that follow nonuniform distributions, such as the Convolutional Neural Networks (CNNs), as described in Section IV-A.

With LNS, operations with higher complexity, such as multiplication, division, square and square root, are easily performed by applying fixed-point addition, subtraction, and left/right bit-shift, respectively. Considering unsigned operands $P$ and $Q$, represented in LNS as $p = \log_2 P$ and $q = \log_2 Q$, and $BitShift(x, z)$, regarded as the left-shifted value of $x$ by an integer $z$ in fixed-point arithmetic, these operations are formulated in (1).

$$
\begin{align*}
\log_2(P \cdot Q) & = p + q; \\
\log_2(P/Q) & = p - q; \\
\log_2(P^2) & = 2 \cdot p = BitShift(p, 1); \\
\log_2(\sqrt{P}) & = \frac{1}{2} \cdot p = BitShift(p, -1).
\end{align*}
$$

This simple logarithmic arithmetic comes at the cost of more complex additions and subtractions, which map in nonlinear operations. For two unsigned numbers $P$ and $Q$ (the sign can be handled in parallel), addition and subtraction can be calculated using Lemma 1.

**Lemma 1.** For $(P, Q) \in \mathbb{N}$, and with Max as the function that returns the maximum value of a tuple of numbers, the following formulation can be adopted to compute addition and subtraction in the LNS domain $(\log_2(P \pm Q))$:

$$
\log_2(|P \pm Q|) = Max(p, q) + \log_2(1 \pm 2^{-|p-q|})
$$

(2)
Proof. For $P \geq Q$, addition and subtraction can be calculated using (3):
\[
\log_2(P \pm Q) = \log_2(P(1 \pm \frac{Q}{P})) = p + \log_2(1 \pm 2^{q-p}) \tag{3}
\]
while for $P < Q$:
\[
\log_2(Q \pm P) = \log_2(Q(1 \pm \frac{P}{Q})) = q + \log_2(1 \pm 2^{p-q}) \tag{4}
\]
By combining (3) and (4):
\[
\log_2(|P \pm Q|) = \max(p, q) + \log_2(1 \pm 2^{-|p-q|}) \tag{5}
\]
and Lemma 1 is proved.

Note that the addition and subtraction operations employ the Gaussian logarithms represented in (6) and in Fig. 4.
\[
G = \log_2(1 \pm 2^\lambda), \quad \lambda = -|p - q| \tag{6}
\]
For a modest number of bits, typically up to 20, the functions in Fig. 4 can be implemented through lookup tables [13]. However, given that the table size increases exponentially with the word length, table-look-up-and-addition methods can be adopted instead to reduce the burden of storing large tables in memory, such as bipartite tables and multipartite methods [14].

For a large number of bits, for example, 32 bits (Fig. 5) or even 64 bits, a piecewise polynomial approximation [15] or the digit-serial methods [17], [18] can be used to implement LNS addition/subtraction. Digit-serial methods, also known as iterative methods, calculate the result digit by digit. Similarly to the COordinate Rotation Digital Computer (CORDIC) method [16], an iterative algorithm can approximate the exponential and logarithmic computation with sequences of digit-serial computations [17]. Signed-digit arithmetic was proposed to simplify the exponential computation and reduce the number of pipeline stages required for LNS addition/subtraction [18]. Although the digit-serial methods require a low circuit area, they exhibit high latency.

The alternative piecewise polynomial approximation methods provide a trade-off between performance and cost. Linear Taylor interpolation has been used to implement 20-bit and 32-bit LNS Arithmetic Units (AUs) by applying a table-based error correction scheme to achieve a similar accuracy to that of their FP counterparts [19]. Although a first-order interpolation is used, the error correction step requires a multiplier, making the complexity similar to that of a second-order interpolator (as shown in Fig. 5 for an LNS AU, for an LNS AU, to be analyzed later in this section). Both Lagrange interpolation [164] and Chebyshev polynomial approximation [20] (second-order approximations) have been used to compute the Gaussian logarithms. Second-order minimax polynomial approximation has also been applied to compute the addition/subtraction in LNS [21].

Higher accuracy than what can be found in standard FP arithmetic is achievable [20].

The approaches referred to above for subtraction in the LNS domain work well, except in the critical region ($\lambda \in [-1, 0]$), due to the singularity of $\log_2(1 - 2^\lambda)$ when $\lambda \to 0$ (see Fig. 4).

![Fig. 4: Plot of the functions $\log_2(1 + 2^\lambda)$ and $\log_2(1 - 2^\lambda)$ (the latter has a singularity when $\lambda \to 0$) between $\lambda = -1$ and $\lambda = 0$). Mathematical transformations, usually known as cotransformations, are used in this region. Using alternative functions defined in adjustable subdomains is one of these transformations. An example of cotransformation is the replacement of the computation of $\log_2(1 - 2^\lambda)$ with two successive subtractions [22], as expressed in (7). In this equation, $2^{k_1}$ is individually chosen for each value, such that the index $\lambda[1]$ for the inner subtraction falls on the nearest modulo-$\Delta[1]$ boundary beneath $\lambda$; $\Delta[1]$ is fixed at a large value, with $\Delta$ representing the interval tables that are stored. The value of the inner product for $\lambda[1]$ can therefore be obtained directly from a lookup table that covers the range $-1 < \lambda < -\Delta[1]$ at modulo-$\Delta[1]$ intervals. Since $-\Delta[1] \leq k[1] < 0$, $2^{k_1} \approx 1$ and the magnitude of the index for the outer subtraction $k_2 < -1$, the computation is shifted away from the critical region [22].

\[
2^p - 2^q = (2^p - 2^q + 2^{k_1}) - 2^{q+k_2} \tag{7}
\]
\[
2^{k_1} + 2^{k_2} = 1 \quad \Rightarrow \quad k_2 = \log_2(1 - 2^{k_1})
\]

A different type of cotransformation decomposes the computation of $G = \log_2(1 \pm 2^\lambda)$ according to (8). The implementation in [23] approximates the cotransformation $\log_2 \left( \frac{1 - 2^\lambda}{1 + 2^\lambda} \right)$ in a limited range [1, 2].

\[
\log_2(1 - 2^\lambda) = \log_2 \left( \frac{1 - 2^\lambda}{1 + 2^\lambda} \right) + \log_2(1 + 2^\lambda) \tag{8}
\]

LNS AUs have been used in the design of configurable and general purpose processors [21], [23]–[26].

The European Logarithmic Microprocessor (ELM) is a 32-bit scalar microprocessor with a fixed-point LNS-based AU ($b = 2$), with numbers represented by a sign bit and a fractional component of 23 bits [26]. This processor implements addition and subtraction by splitting the range of $\lambda$ into segments for every increasing power of 2. In turn, these segments are split into smaller intervals, and the function $F = \log_2(1 + 2^\lambda)$ and its derivative ($D$) are stored in memory for each of those intervals, as depicted in Fig. 5 with the values of intermediate points estimated with a Taylor interpolation. To calculate the error for each interval, the maximum error $E$ is tabulated alongside $F$ and $D$ – this error corresponds to when a point falls near the next stored value tuple. A separate table of
The selection of proportions \((P)\) stores the normalized shape of the common error curve. Each proportion value will be 0 - when the required value is at the beginning of a segment - or up to 1 - when it is estimated by the interpolation. The error is calculated by adding to the result of the interpolation the value of \(E \times P\). In Fig. 5 a range shifter is inserted immediately after the selection of \(p\) for the calculation of \(\lambda\). If \(\lambda\) falls close to zero in the subtraction operation, \(p\) and \(\lambda\) will be transformed into new values, and \(\lambda\) will be moved to the linear region by applying the cotransformation (7) [15].

The ELM adopts a register-memory addressing mode Instruction Set Architecture (ISA), with a single-level cache integrated into the pipeline. The architecture includes 16 general-purpose registers, an 8 kbyte \(L1\) data cache, two adders/subtractors operating in three clock cycles, and four combined multiplier/divider/integer units operating in one clock cycle. Vector operations are implemented in the ELM by using four identical functional units in parallel, requiring a data cache organization in which four successive words are read out simultaneously. The ELM is designed and fabricated in 0.18\(\mu\)m technology [26]. Its performance and accuracy, running at 125 MHz, are evaluated against the TMS320C6711 contemporary Digital Signal Processor (DSP) [27], available in the same technology and running at 150 MHz. The TMS320C6711 is a Very Long Instruction Word (VLIW) processor that is able to issue eight independent instructions issued every cycle with two pipelined FP adders and two other pipelined FP multipliers with a latency of four cycles and two integer units with a latency of one cycle. It has been shown that the ELM is marginally more accurate than the FP processor for additive operators, while multiplicative operators return exact results. For a typical kernel, making equal use of each one of the operators, LNS will incur roughly half the total error of FP arithmetic. Moreover, notwithstanding the ELM running at \(\frac{5}{6}\) of the clock speed of the FP processor, the additive times were marginally better, the multiplications were 3.4 times faster, and the divisions and square roots were many times faster in the former than in the latter processor. Although LNS enables the design of faster and, typically, more accurate processors than those of FP arithmetic (the LNS implementation tends to have smaller worst-case relative errors than those of FP [28]), general purpose processors currently still adopt the standardized FP, which also easily interfaces with integer binary arithmetic.

Another LNS-based AU that adopts the hardware-efficient cotransformation [8] for subtraction in the critical region, computing the second term of this equation with a second-order polynomial, was proposed in [23]. The architecture of the LNS-based AU consists of four main blocks: \(i\) the pre- and postprocessing blocks, including the preparation steps of the operations and the combination and/or selection of the results produced by the two main interpolation blocks; \(ii\) the main interpolator block that implements the approximations for computing \(\log_2(1 + 2^i)\) and exponentiation on the complete range \((-25, 0]\), and \(\log_2(1 - 2^i)\) outside the critical region \((-25, -0.25]\), using second-order piecewise polynomial approximations; and \(iii\) the critical region block used to calculate the logarithm of the cotransformation function (8) for the critical region \((-0.25, 0]\) by approximating the cotransformation function \(1 - \frac{2^i}{1 + 2^i}\) using a first-order polynomial and computing the required \(\log_2\) function based on a range reduction technique that normalizes the function argument to the range \([1, 2]\). Since only one instruction is evaluated at a given time, interpolator data paths within the two parallel blocks \(i\) and \(ii\) can be shared to obtain a more compact design.

The LNS-based AU proposed in [23] was integrated into the datapath of a custom 32-bit OpenRISC core [29] using a CMOS 65 nm process. For comparison purposes, an identical system was designed with featuring a standard IEEE 754 single-precision FP AUs with hardware support for additions, subtractions and multiplications. The error analysis show similar maximum and average values to the ones achieved for addition and subtraction with the ELM. Executing also at a clock frequency of 125 MHz, the adder and subtractor have a latency of four clock cycles, while the multiplier, divisor and square root calculus exhibit a latency of one clock cycle. The LNS-based adder/subtractor of this processor significantly reduces the circuit area required by the individual LNS units, mainly due to circuit sharing. Comparing the energy efficiency of the LNS (per LOGarithmic Operation (LOGP)) and the FP-based AUs (per FLoating-Point Operation (FLOP)) for a CMOS 65 nm process, addition and subtraction in FP is approximately 3 times more efficient than in LNS (40 pJ/FLOP vs 130 pJ/LOGP), while multiplication in LNS is approximately 1.5 times more efficient (48 pJ/FLOP vs 30 pJ/LOGP), division
is 17 times more efficient (525 pJ/FLOP vs 30 pJ/LOGP), and square root is 38 times more efficient than FP (609 pJ/FLOP vs 16 pJ/LOGP).

A combination of the LNS and FP representations have been explored as an approach to designing hybrid AUs [30]. It enables, for example, the multiply and divide operations to be computed using the LNS format, while addition and subtraction are efficiently performed in FP. The hardware for performing the FP-to-LNS and LNS-to-FP conversions is embedded within the hybrid AUs.

B. Residue number systems

The Chinese Remainder Theorem (CRT) was originally proposed in the 3rd century AD [31]. In the 1950s, it became the support for the proposal of computer arithmetic based on the RNS [32]. The CRT states that given a moduli set \( \{m_1, m_2, \ldots, m_n\} \) of pairwise co-prime numbers, i.e., the Greatest Common Divisor (GCD)(\(m_i, m_j\)) = 1 for \( i \neq j \) \((\langle X \rangle_m \) represents the residue of \( X \) for the modulus \( m \)):

\[
\langle X \rangle_m \ | 1 \leq i \leq n \} \text{ is unique for any } 0 \leq X < M ; \quad M = \prod_{i=1}^{n} m_i \tag{9}
\]

The CRT asserts that there is a bijection, formally described in Lemma 2.

**Lemma 2.** Let \( m_1, \ldots, m_n \) be \( n \) pairwise co-prime numbers and \( M = \prod_{i=1}^{n} m_i \). The CRT asserts that there is a bijection between \( \mathbb{Z}_M \) and \( \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n} \):

\[
\mathbb{Z}_M \leftrightarrow \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n} \tag{10}
\]

**Proof.** Consider \( n = 2 \) and two co-prime numbers \( m_1 \) and \( m_2 \). The modular inverse \( m_1^{-1} = \langle m_1^{-1} \rangle_{m_2} \) and \( m_2^{-1} = \langle m_2^{-1} \rangle_{m_1} \) exist since \( m_1 \) and \( m_2 \) are co-prime numbers [165]. Thus, the value \( Y \) satisfies the equation for \( X \):

\[
Y = \langle x_1 m_2 m_2^{-1} + x_2 m_1 m_1^{-1} \rangle_M = m_1 \times m_2
\]

since \( \langle Y \rangle_{m_1} = x_1 \) and \( \langle Y \rangle_{m_2} = x_2 \).

It remains to be shown that no other solutions exist modulo \( M \), which means \( X = Y \pmod{M} \). By contradiction, it is assumed that \( \langle X \rangle \neq \langle Y \rangle \) and \( \langle X \rangle_{m_i} = \langle Y \rangle_{m_i} \), which means \( \langle X - Y \rangle_{m_i} = 0 \), \( i = 1, 2 \). Hence, \( \langle X - Y \rangle \) is the Least Common Multiple (LCM) of \( m_i \). However, this contradicts the fact that \( m_i \) are pairwise co-prime; thus, the LCM is \( M \). We can represent an element of \( \mathbb{Z}_M \) by one element of \( \mathbb{Z}_{m_1} \) and one element of \( \mathbb{Z}_{m_2} \), and vice versa.

The CRT can now be proven for an arbitrary number \( n \) (number of co-prime integers) \( m_i \leq i \leq n \) by induction. Defining \( M_i = \frac{M}{m_i} \) and \( M_i^{-1} = \langle M_i \rangle_{m_i} \), the previous equation for \( Y \) becomes:

\[
\langle X \rangle = \prod_{i=1}^{n} x_i \times M_i \times M_i^{-1} \tag{n}
\]

and \( X \) is the unique solution in the ring \( \mathbb{Z}_M \).

Moreover, for \( m_1, \ldots, m_n \) pairwise co-prime integers and \( M = m_1 \times \cdots \times m_n \), the CRT also asserts the following ring isomorphism [33]:

\[
\mathbb{Z}_M \rightarrow \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n}
\]

\[
X \rightarrow f(X) = (x_1, \ldots, x_n)
\]

\[
= (\langle X \rangle_{m_1}, \ldots, \langle X \rangle_{m_n}) \tag{11}
\]

which allows carrying the ring operations on the residues for each modulo independently.

Thus, RNS is a nonpositional number system based on smaller residues, the quantity and range of which are controlled by the chosen pairwise co-prime moduli set [34]. Each residue corresponds to a "digit", and "digits" can be processed in parallel. The direct application of RNS properties to addition and multiplication expressed in (12) is widely disseminated in the design of efficient arithmetic, in particular for linear processing [35], [36].

\[
X \circ Y \stackrel{0,+,-,\ast \Rightarrow}{\rightarrow} \{\langle x_1 \circ y_1 \rangle_{m_1}, \ldots, \langle x_n \circ y_n \rangle_{m_n}\} \tag{12}
\]

RNS has been proposed mainly for fixed-point arithmetic, accommodating also numbers with signs in the range \([-\frac{M}{2} : \frac{M}{2})\]. The main limitation of RNS-based arithmetic is related to the operations, such as division, comparison, sign and overflow detection, which are difficult to implement with non-weighted number representations. These require a combination of the residues to reach some type of positional representation, i.e., the partial or complete application of the RNS-to-binary (reverse) conversion, supported by the CRT or the Mixed Radix Conversion (MRC) [37], [38]. Although both approaches use modular operations, the CRT carries out the computation in a single step over a large modular sum \( X (M_i = \langle M \rangle_{m_i}) \) is the multiplicative inverse of \( M_i \), and \( \alpha \) is an integer [13].

\[
X = \sum_{i=1}^{n} M_i \langle M_i^{-1} \rangle_{m_i} x_i = \sum_{i=1}^{n} M_i \langle M_i^{-1} \rangle_{m_i} x_i - \alpha M \tag{13}
\]

On the other hand, the MRC uses shorter modulo arithmetic with an iterative procedure:

\[
X = y_1 + m_1 \times (y_2 + m_2 \times (y_3 + m_3 \times (\cdots))) \tag{14}
\]

\[
y_1 = x_1
\]

\[
y_2 = (x_2 - y_1) \times \langle m_1^{-1} \rangle_{m_2}
\]

\[
y_3 = (x_3 - y_1) \times \langle m_1^{-1} \rangle_{m_3} - y_2 \times \langle m_2^{-1} \rangle_{m_3}
\]

\[
\cdots
\]

Algorithms for RNS division [39], comparison [40], [41], sign detection [42], scaling [43], and reduction [44] have been proposed for specific moduli sets. Basis Extension (BE) is often used in those algorithms to extend the representation of an RNS moduli set (often called a basis) \( S_1 \) to another \( S_2 \). Consider the two bases \( S_1 = \{m_{1,1}, \ldots, m_{1,n_1}\} \) and \( S_2 = \{m_{2,1}, \ldots, m_{2,n_2}\} \) with dynamic ranges \( M_i = \prod_{i=1}^{n_i} m_{i,i} \) and
\[ M_2 = \prod_{i=1}^{n_2} m_{2,i}, \text{ respectively. If an error of } \alpha M_1 \text{ is tolerated, the residues associated with the second basis can be computed as in (14)}: \]
\[ x_{2,i} = \langle X' \rangle_{m_{2,i}} = \left\langle \sum_{i=1}^{n_1} M_i \langle M_i^{-1} \rangle_{m_{1,i}} x_i \right\rangle_{m_{2,i}} \quad (15) \]

For applications that do not tolerate this error, a method to perform the exact extension requires an extra modulus \( (m_e) \), with the corresponding residue \( x_e = \langle X \rangle_{m_e} \), to enable the computation of a parameter \( \alpha \), for \( X < \delta M_2, \delta \in \mathbb{N} \), and \( x_e \geq 2(\delta + n_2 + 1) \) [166]:
\[ \alpha = \left\langle (X' - x_e) \langle M_i^{-1} \rangle_{m_i} \right\rangle_{m_e} \quad (16) \]

Once \( \alpha \) is known, the basis extension is finally computed with (17).
\[ x_{2,i} = \left\langle \sum_{i=1}^{n_1} M_i \langle M_i^{-1} \rangle_{m_{1,i}} x_i - \alpha M_1 \right\rangle_{m_{2,i}} \quad (17) \]

Performing efficient RNS arithmetic requires the selection of the most suitable moduli set for the problem at hand. The size of the set and the width of the moduli are mainly defined by the dynamic range and the desired parallelism. Features usually considered to select the moduli are i) the efficiency of the modular arithmetic for each channel (the term “channel” is usually adopted to designate a residue and the modular arithmetic units associated with it), which typically leads to values near a power of two; ii) the balancing of the moduli set, since the slowest channel defines the overall performance; and iii) the cost of the reverse converters, with some moduli sets allowing efficient arithmetic channels at the cost of more complex reverse converters and others tailored to the result in simple reverse conversion circuits due to the mathematical relations between the moduli [45]. Altogether, the most used moduli sets are composed of modulus \( 2^n \pm k \), with \( v \in \{ n, 2n \} \) and \( k \in \{-1, 0, +1\} \) [35].

Alternative binary codings can be explored for modular arithmetic in each RNS channel. Thermometer Coding (TC) and One-Hot Coding (OHC) are among some of the most used ones [46]. The value of a number in TC is expressed by the number of 1’s in the string of bits [47], which means that it is a nonpositional redundant number representation. Typically, for simplicity, a run-length of 1’s is placed at one end of the string. For the OHC, the only valid combinations in a string of bits are those with only a single bit ‘1’ and all the others ‘0’. For the OHC, the bit ‘1’ position directly indicates the value of the number. \( k \) and \( k + 1 \) bits are required to represent integers between 0 and \( k \) in TC and OHC, respectively. For example, for \( k = 7 \), the number 4 can be represented in TC by the stream “0001111”, while in the OHC, it can be represented by the combination “00010000”. The TC is often used in Digital-to-Analog Converters (DACs), as it leads to less glitching than binary-weighted codings. The OHC is used in Finite State Machines (FSMs), as it avoids the use of decoders: a single ‘1’ bit directly identifies the state. The TC and OHC codings are most advantageous for RNS moduli that result in small and medium dynamic ranges [46]–[48].

Several RNS-based processors have been developed, for example, a fully programmable RISC Digital Signal Processor (RDSP) [167] and hardware/software RNS-based units [49]. The RDSP is a 5-stage pipeline processor with the RNS-based AU partially located in the third and fourth pipeline stages, as depicted in Fig. 6 The most popular 3-moduli set \( \{2^n - 1, 2^n, 2^{n+1} + 1\} \) is balanced by extending the binary channel to \( 2^{2n} \); the reverse converter is distributed by the two aforementioned pipeline stages, while, for the multiply-accumulate (MAC) unit, the multiplier is located in the third stage, and the accumulator operates in parallel to the load/store units in the fourth pipeline stage. The experimental results obtained for the RDSP in the CMOS 250 nm technology show that not only are the circuit area and the power consumption significantly lower but also a speedup is obtained in comparison to a similar DSP using a data path based on binary arithmetic.

An AU based on a moduli set that comprises a larger modulus \( 2^k \) and all the other moduli of type \( 2^n - 1 \) assumes that the selected moduli fit the typical word size of a Central Processing Unit (CPU), for example, 64 bits [49]. Adopting a hardware-software approach, RNS adders, subtractors and multipliers are implemented in the hardware, while the reverse conversion and nonlinear operations are implemented in the software. A minimum set of instructions includes not only forward conversion (residue), modular addition (add_m) and modular multiplication (mult_m) but also instructions that operate on positional number representations, such as add, sub, AND and SHR (logical shift right). It supports changing the dynamic range at runtime in a simple manner, which in turn can result in a reduction in both power consumption and execution time.

Although the RNS was proposed mainly for fixed-point arithmetic, it has also been used for floating-point units [50], [51]. To operate on FP numbers, the mantissa and the exponent are individually converted and processed in the RNS domain. For example, to multiply two FP numbers, both mantissas are multiplied and exponents are added in the RNS domain.

A variant of RNS, the Redundant Residue Number System (RRNS) addresses different purposes [52] but is mainly used for error detection and correction. Since the residues are independent of each other, introducing redundant moduli in the set of original (information) moduli, usually called the legitimate, the representation can be extended to the illegitimate range to detect residue errors and, in some cases, correct them. If a result falls into the illegitimate range, it can be concluded that i) one or more residue digit errors exist as long as the number of residue digit errors is not more than twice the number of redundant moduli and ii) errors can be corrected by subtracting the error digits from the residue digits, assuming that the number of residue errors does not exceed the number of redundant moduli [35]. For example, applying the RRNS 3-moduli set \( \{2^n - 1, 2^n + 1, 2^{n+1}\} \) and considering the legitimate range \( [0, 2^{2n} - 2] \), from the moduli subset \( \{2^n - 1, 2^n + 1\} \), a one-digit error can be detected in the illegitimate range \( [2^{2n} - 1, 2^{3n+1} - 2^{n+1} - 1] \). It is
worth noting that the independence of residue digits in RRNS prevents a residue error in one channel from propagating to another.

A computationally redundant processing core with high energy efficiency has been proposed [53]. The computationally redundant energy-efficient processing for y’all (CREEPY) core comprises microarchitecture-, ISA- and RRNS-centered algorithms and techniques to efficiently perform computational error correction. This work has shown significant improvements over a non-error-correcting binary processing core. Although all research on the design of cores based on the RRNS, the microarchitectures abstract away the memory hierarchy and do not consider the power-performance impact of RNS-based memory addressing. Compared with a non-error correcting core that addresses memory in binary, direct RNS-based memory addressing schemes slow down in-order/out-of-order cores. Novel schemes for RNS-based memory access, extending low-power and energy-efficient RRNS-based architectures, are proposed in [54].

C. Stochastic number representation

A stochastic signal is the result of a continuous-time stochastic process with two possible values, represented by the symbols 0 and 1 [55]. Numbers are represented by probabilities associated with bitstreams: $p_i$ is the value of a stochastic bitstream $I_i$ defined as the ratio of 1 bits to the total number of bits. Considering Unipolar Representation (UR), the stochastic bitstream values are bound $[0 : 1]$, while for Bipolar Representation (BR), the values are bound by $[-1 : 1]$, a result of applying the scale factor 2 after a negative bias term (18).

For example, a stream $I$ containing 60% 1s and 40% 0s, as depicted in Fig. 7, represents the numbers 0.6 in the UR and 0.2 in the BR.

$$UR = p(i) \quad BR = 2 \times (p(i) - 0.5) \quad (18)$$

Neither the length or the structure of $I$ needs to be fixed. The number represented by a particular stream is not defined by the individual value of a bit in a specific position; thus, SC relies on a nonpositional number representation. The randomization of the representation, associated with the mapping from space to time, leads to SC arithmetic that can be implemented with simple logic circuits [56], [57].

For example, a single two-input logic gate can be used to compute the product of the probabilities $p \times q$ (Fig. 8). It is easy to show that for the UR, it is an AND gate, while for the BR, it is an eXclusive-NOR (XNOR) gate. A 2 : 1-bit multiplexer can be used to add two numbers in UR, as depicted in Fig 8B.

The addition is scaled by $\frac{1}{2}$, as shown in (19).

$$p(O) = p(I_1)p(I_3) + (1 - p(I_3))p(I_2)$$
$$p(I_3) = \frac{1}{2}$$
$$p(O) = \frac{1}{2} \times (p(I_1) + p(I_2)) \quad (19)$$
The correctness of the final result is impacted by the correlation, or lack of randomness, between stochastic bitstreams. For example, in the extreme case where the same bitstream feeds the two inputs of the AND gate in Fig. 8a, regardless of the bit values in the bitstream, for UR, \( p(O) = p(I_1) = p(I_2) \) instead of \( p(O) = p(I_1)^2 = p(I_2)^2 \). For the same bitstream feeding the two entries of the XNOR gate in Fig. 8c, the value of the product is \( p(O) = 1 \) for the BR.

Hence, it is important to use “good” random generators. However, even in systems where stochastic bitstreams are generated with high-quality random generators, after a few operations, the bitstreams tend to become correlated due to clock synchronicity. One way to overcome this limitation is to regenerate the bitstream [59]; however, the resource penalty impacts the benefit introduced by the compact arithmetic operators. A less costly alternative is to replace the global clock with individual ring oscillators for each synchronous component, which can be sensitive to the process, temperature and voltage variation [58].

To use SC in typical computing systems, binary-to-stochastic and stochastic-to-binary converters are required to leverage SC in practical applications. These converters are presented in Fig. 9. The binary-to-stochastic conversion in Fig. 9a comprises the generation of an \( m \)-bit random binary number in each clock cycle by means of comparing the \( m \)-bit input binary number with the output of a (pseudo)random number generator. For UR, assuming that the random numbers are uniformly distributed over the interval \([0, 1]\), the probability of a 1 being generated by the comparator for a given clock cycle is equal to the fractional number at the input of the converter. For the stochastic-to-binary conversion in Fig. 9b, the value \( p \) is carried by the number of 1s in its bitstream; thus, the conversion is performed by counting the numbers of 1s to obtain \( p \) and normalizing the result by the number of bits in the stream.

Arithmetic functions, even nonpolynomial functions using a MacLaurin expansion [60], can be implemented with SC with simple combinatorial elements. However, highly nonlinear functions such as the \( \tanh \) and \( \max \) functions, widely used in artificial neural networks as discussed in Section IV-A require FSM-based SC elements [61].

In the state machine in Fig. 10, states are organized in sequence, as in a saturating counter [62]. The number of states is given by \( N = 2^n \); thus, the set of states can be implemented by \( n \) flip-flops. \( X \) is the input of the state machine, and \( Y \) is the output (not shown in this figure), with the latter depending only on the state. Assuming that the input \( X \) is a stochastic bitstream, typically a Bernoulli sequence, the probability \( p(X) \) of a bit in the sequence being ‘1’ follows a binomial distribution. With \( p(Y) \), the probability of a bit in the output stream \( Y \) being ‘1’, and \( p_i \), the probability of the state \( S_i \) given the input probability \( p(X) \), (22) can be derived. (22) states that the probability of transitioning from \( S_{i-1} \) to \( S_i \) must equal the probability of transitioning from \( S_i \) to \( S_{i-1} \) and that the sum of \( p_i \) for all \( N \) states should be unitary. In (22), \( s_i \) takes the value 1 if the output \( Y = 1 \) in the current state \( S_i \) and 0 otherwise.

\[
p_i(1 - p(X)) = p_{i-1}p(X)
\]

(20)

\[
\sum_{i=0}^{N-1} p_i = 1
\]

(21)

\[
p(Y) = \sum_{i=0}^{N-1} s_i p_i
\]

(22)

Lemma 3 enunciates the value \( s_i \) such that \( \tanh \) can be computed with SC.

**Lemma 3.** By inserting \( s_i \) as in (23) into (22):

\[
s_i = \begin{cases} 0, & 0 \leq i < N/2 \\ 1, & N/2 \leq i < N \
\end{cases}
\]

(23)

tanh is computed based on SC with (24):

\[
z = \tanh\left(\frac{N}{2}x\right)
\]

(24)

**Proof.** Starting from (20), assuming \( 0 < p_i < 1 \) and \( p_0 = 1 \), it is easy to derive (25):

\[
p_i = \left(\frac{p(X)}{1 - p(X)}\right)^i
\]

(25)

and by applying (21) to normalize (25), we obtain (26):

\[
p_i = \frac{p(X)}{1 - p(X)}
\]

(26)

\[
\sum_{j=0}^{N-1} \left(\frac{p(X)}{1 - p(X)}\right)^j
\]

Based on the configuration in (23), (22) can be rewritten as (27):

\[
p(Z) = \sum_{i=N/2}^{N-1} s_i p_i
\]

(27)
and by substituting (26) in (27), we obtain (28):

$$p(Z) = \sum_{i=N/2}^{N-1} \left( \frac{1}{1 - p(X)} \right)^i \left( \frac{p(X)}{1 - p(X)} \right)^j = \left( \frac{p(X)}{1 - p(X)} \right)^{N/2} \left( 1 - \left( \frac{p(X)}{1 - p(X)} \right)^N \right)$$

Adopting the bipolar coding format, which means that the ranges of real numbers $x$ and $z$ are extended to $1 \leq x, z \leq 1$, and assuming that the probability of a bit in the stream being one $P(\alpha = 1) = \frac{1 + z}{2}$, (28) leads to (29):

$$z = \frac{1 + x}{2} = \frac{1 + z}{2}$$

By using the Taylor expansion shown in (30), (29) is rewritten to compute tanh based on SC (31):

$$1 + x \approx e^x; \quad 1 - x \approx e^{-x}$$

$$z = \frac{e^{N/2} x - e^{-N/2} x}{e^{N/2} x + e^{-N/2} x} = \tanh(\frac{N}{2} x)$$

and Lemma 3 is proved.

The stochastic implementation of other functions, such as the absolute value, the exponentiation, and the max/min functions can be carried out using linear FSMs [63] and [61]. These functions support, for example, the development of deep learning applications, as described in Section IV-A.

The characteristics of SC make it suitable, in particular, for designing embedded systems. It offers a lower circuit area and power consumption, as well as high error resilience, in comparison to their binary system counterparts [64].

Since operations at the logic level are performed on randomized values, an attractive feature of SC is the degree of tolerance to errors [65]. A stochastic bitstream of length $m$ represents numbers with a precision of $1/m$. The error follows a normal distribution with zero mean and a variance that is inversely proportional to the bitstream length $N$ ($\sigma \approx \frac{1}{\sqrt{N}}$) [56]. For example, insufficient bitstream length or randomness can be leveraged in approximate computing [66], [67].

Several processors, more or less dedicated, have been developed to apply SC to different applications, such as Artificial Neural Networks (ANNs), control systems, and the decoding of modern error-correcting codes [67]. Stochastic Recognition and Mining processor (StoRM) uses SC to efficiently materialize computational kernels to perform classification, search, and inference on real data [68]. The StoRM architecture is a 2D array of stochastic Processing Elements (PEs) (typically $15 \times 15$) with a streaming memory hierarchy, mitigating the overhead of binary-to-stochastic conversion by sharing the corresponding arithmetic units across rows or columns of stochastic PEs. Experimental results from the implementation of StoRM in CMOS TSMC 65 nm technology show that SC allows a highly compact implementation, resulting in order-of-magnitude less circuit area and power consumption.

However, computation with SC requires multiple cycles, equal to the number of bits in the stochastic bitstream. Since the length of the bitstream grows with the precision of the data being represented, regardless of its lower power consumption, the PE ends up consuming more energy than a conventional binary processing element. To mitigate this intrinsic limitation (increase in cycle count) of SC, vector processing and segmented stochastic processing are adopted [68]. Segmented stochastic processing is a hybrid representation that divides the binary data into equally sized segments that are represented by individual stochastic bitstreams so as to achieve a trade-off between precision and bitstream length.

The following sections show how SC can be applied to design invertible logic on the current CMOS technology, based on Boltzmann machines [78], and to homomorphic encryption.

D. Hyperdimensional representation and arithmetic

The circuits of the brain are composed of billions of neurons, with each neuron typically connected to up to 10,000 other neurons. This situation leads to more than 100 trillion modifiable synapses [69]. High-dimensional modeling of neural circuits not only supports artificial neural networks but also inspires parallel distributed processing. It explores the properties of high-dimensional spaces, with applications in, for example, classification, pattern identification and discrimination.

HDC, inspired by the brain-like computing paradigm, is supported by random high-dimensional vectors [70]. It looks at computing with very wide words, represented by high-dimensional binary vectors, which are typically called hyperdimensional vectors, with a dimensionality on the order of thousands of bits. It is an alternative to Support Vector Machines (SVMs) and CNN-based approaches for supervised classification. With Associative Memory (AM), a pattern $X$ can be stored using another pattern $A$ as the address, and $X$ can be retrieved later from the memory address $A$. However, $X$ can also be retrieved by addressing the memory with a pattern $A'$ similar to $A$.

The high number of bits in HDC are not used to improve the precision of the information to represent, as it would be difficult to give an exact meaning to a 10,000-bit vector in the same way a traditional computing model does. Instead, they ensure robustness and randomness by (i) being tolerant to errors and component failure, which come from redundancy in the representation (many patterns mean the same thing and thus are considered equivalent) and (ii) having highly structured information, as in the brain, to deal with the arbitrariness of the neural code.

HDC starts with vectors drawn randomly from hyperspace and uses AM to store and access them. A space of $n = 10,000$-dimensional vectors and independent and identically distributed (i.i.d.) components drawn from a normal distribution with mean $\mu = 0$ can be considered as a typical
example. Points in the space can be viewed as corners of a 10,000-dimensional unit hypercube, for which the distance \(d(X,Y)\) between two vectors \(X,Y\) is expressed using the Hamming metric. It corresponds to the length of the shortest path between the corner points along the edges (the distance is often normalized to the number of dimensions so that the maximum distance, when the values of all bits differ, is 1). Assuming that the distance \((k)\) from any point of the space to a randomly drawn point follows a binomial distribution, with \(p(0) = 0.5\) and \(p(1) = 0.5\):

\[
f(k) = \binom{10,000}{k} 0.5^k 0.5^{10,000-k} = \frac{10,000!}{k!(10,000 - k)!} 0.5^{10,000} (32)
\]

with mean \(\mu = 10,000 \times 0.5 = 5,000\) and variance \(VAR = 10,000 \times 0.5 \times 0.5 = 2,500\) (the standard deviation \(\sigma = 50\)). This means that if vectors are randomly selected, they differ by approximately 5,000 bits, with a normalized distance of 0.5. These vectors are therefore considered “unrelated” [70]. Although it is intuitive that half the space has a normalized distance from a point of less than 0.5, this statement is not true if we observe that it is only less than a thousand-millionth closer than 0.47 and another thousand-millionth farther than 0.53. This distribution provides robustness to the hyperdimensional space, given that “nearly” all of the space is concentrated around the 5,000 mean distance (0.5). Hence, a 10,000-bit vector representing an entity may see a large number of bits, e.g., a third, changing their values, by errors or noise, and the resulting vector still identifies the correct one because it is far from any “unrelated” vector.

The vector representation of patterns enables the use of the body of knowledge on vector and matrix algebra to implement hyperdimensional arithmetic. For example, the componentwise addition of a set of vectors results in a vector with the same dimensionality that may represent that set. The sum-vector values can be normalized to yield a mean vector. Moreover, a binary vector can be obtained by applying a threshold. If there are no duplicated elements in a set, the sum-vector is a possible representative of the set that makes up the sum [70].

Another important arithmetic operation in HDC is vector multiplication, which, as for SC with BR, can be implemented with the bitwise logic XOR operator (see Fig. 8c). Reverting the order of the BR from \((1, -1)\) to \((0, 1)\), the ordinary multiplication of binary vectors \(X\) and \(Y\) can be implemented by the bitwise eXclusive-OR (XOR). It is easy to show that XOR is commutative and that each vector is its own inverse \((X * X = 0)\), where \(0\) is the unit vector \((X * 0 = X)\). Multiplication can be considered as a mapping of points in the space: multiplying vector \(X\) by \(M\) maps the vector into a new vector \(X_M\) as far from \(X\) as the number of 1s in \(M\) (represented by \(|M|\)) (35). If \(M\) is a random vector, with half of the bits taking, on average, a value 1, \(X_M\) is “unrelated” to \(X\), and it can be said that multiplication randomizes it.

\[
d(X_M, X) = |X_M * X| = |M * X * X| = |M| (33)
\]

Lemma 4 shows that multiplication is distributive over addition, and it implements a mapping that preserves distance. These are useful properties of hyperdimensional arithmetic. By keeping the relation between the vectors, multiplication preserves the relation between entities.

**Lemma 4.** Multiplication is distributive over addition and implements a mapping that preserves distance, i.e., mapping two vectors \((X, Y)\) with the same vector \(M\) in the hyperdimensional space yields:

\[
d(X_M, Y_M) = d(X, Y) \quad (34)
\]

**Proof.** If the bipolar \((-1, 1)\) representation is adopted, the XOR operator is replaced by regular multiplication. Since it determines over ordinary addition, it also does so in this bipolar case. The symbol \(| \cdot |\) in (35) denotes normalization.

\[
M * |X + Y| = |M * X + M * Y| (35)
\]

By mapping two vectors \((X, Y)\) with the same vector \(M\), the relation between the vectors is maintained as shown in (36):

\[
X_M * Y_M = (M * X) * (M * Y) = X * Y \quad |X_M * Y_M| \Rightarrow d(X_M, Y_M) = d(X, Y) \quad (36)
\]

and Lemma 4 is proved.

**Permutation** changes the order of vector components. The permutation can be described as a list of integers \((1, 2, 3, \cdots, 10,000)\) in the permuted order or represented as multiplication by a special kind of matrix. This permutation matrix will have all ‘0’s except exactly one ‘1’ in each row and each column. Similar to vector multiplication, when a permutation is applied to different vectors, the distance between them is maintained. As with sets, elements of a sequence can be represented by a single hypervector in a process called flattening. However, if sequences are flattened by the sum, the order of the elements will be lost. Thus, the elements must be “labeled” according to their position in the sequence so that the value of \(X\) appears under different variants depending on the position in the sequence, which can be done with permutations [70]. Sequences in time are important to introduce memory in the systems, which can also be represented with pointer chains or linked lists [70].

The HDC has several known applications. For example, sequence prediction based on sparse hyperdimensional coding is used for online learning and prediction [71], which can be useful for predicting mobile phone use patterns, including the prediction of the next launched application and the next GPS location of a user. Another example of an HDC application is the learning and classification of biosignals, such as electromyography (EMG), electroencephalography (EEG), and electrocorticography (ECoG) [72]. A full set of Hyper-Dimensional (HD) network templates comprehensively encodes body potentials and brain neural activity recorded from different electrodes within a single HD, processed as a single entity for learning and robust classification purposes.

The diagram of a programmable processor based on HDC for supervised classification [73] is presented in Fig. [11] This HDC processor encompasses three main components, each corresponding to processing steps: the Item Memory (IM),
the Encoder and the AM. The IM stores a sufficiently large collection of random hypervectors called items. This memory maps symbols to items in the inference phase, as it was learned in the training phase. The Data Processing Units (DPUs) of the Encoder combines the input hypervector sequence according to the application-specific algorithm to compose a single hypervector for each class. Finally, the AM stores the trained class of hypervectors and delivers the best prediction according to the Hamming distance (\(d_h\)). The inputs of the processor comprise the assignment of symbols to an item in the IM, and the outputs comprise the assignment of class labels to the AM address, all of which is enabled by the HD mapper.

An Application Specific Integrated Circuit (ASIC) for the HDC-based programmable processor, with 2048-bit vectors, was synthesized and implemented using a 28 nm process with a high-k dielectric coupled with metal gate electrodes (HK/MG) [73]. The chip showcases high energy efficiency, less than 1.5 J/prediction for a comprehensive benchmark at clock cycle of approximately 2.5 ns. It rivals other state-of-the-art processors targeting supervised learning, in terms of both accuracy and energy efficiency.

E. Discussion and approximate arithmetic techniques

The LNS swaps the cost of addition/subtraction with the cost of the multiplication/division operations, as well as powers and roots, but it still leads to a large cost difference between these two operations categories. Although LNSs provide a similar range and precision to those of FP, beyond the simplification of multiplication and division to fixed-point addition and subtraction, the FP number systems have become a standard, while the LNS is mainly applied in niches. This occurs, for example, in signal processing, most recently in machine learning, as shown in the next sections.

RNS belongs to the class of nonpositional representations exposing parallelism at a very fundamental computing level to achieve high performance and at the same time supports the design of reliable systems (fault tolerance). Although the usage of RNS allows the design of more energy-efficient systems, RNS does not guarantee a decrement in cost or power consumption, since the reduction operation has to be applied and the sum of the residue bit-widths is typically the bit-width of the equivalent fixed-point binary number. It can be applied not only to designing hardware devices but also to speeding up the execution of applications, such as in cryptography, on traditional binary-based programmable systems (e.g., the massively parallel Graphic Processing Units (GPUs)). However, RNS is not suitable for designing comparison units, dividers and other nonlinear operations.

HDC is inspired by the attributes of neuronal circuits, including hyperdimensionality and (pseudo)randomness. When employed on tasks such as learning and classification, HDC involves the manipulation and comparison of large patterns (typically 10,000-bit patterns) within memory. It allows systems to retain memory, instead of computing everything they sense, pointing more to a kind of Processing In-Memory (PIM). The representation of the patterns, and the associated arithmetic, does not assign any weight to the “digits” according to their position in the pattern; thus, it is also a nonpositional representation. A key attribute of hyperdimensional computing is its robustness to the imperfections associated with the technology on which it is implemented. HDC improves with the advances in memory technologies, replacing computation with the manipulation of patterns in memory.

SC has shown promising results for low-power area-efficient hardware implementations, even though existing stochastic algorithms require long streams that negatively impact the latency. Being another nonpositional representation, SC improves the robustness of circuits subject to random faults or component variability. Since with SC, multiplications and additions can be calculated using AND gates and multiplexers, significant reductions in power and the hardware footprint can be achieved in comparison to the conventional binary arithmetic implementations. The significant savings in power consumption and hardware resources allow the design of autonomous systems, such as embedded devices or for computation on the edge.

SC is a good example of a representation that easily allows trading off the computation quality with the performance and energy consumption by adjusting the bitstream length. The idea of trading off accuracy with performance and energy efficiency led to the approximate computing paradigm for designing circuits and systems [74]. This paradigm has been explored at several levels, from basic circuits to components, processing units, and programs. In particular, approximate arithmetic circuits have been proposed, including adders, multipliers and dividers [75]. Both error characteristics, such as the “error rate” and the “error distance”, and circuit measurements, such as the delay, cost and power dissipation, should be considered for approximate circuits [75]. Approximate computing has been considered for several applications with error resilience, such as image processing and machine learning [76, 77]. Although most of the research on approximate computing up to now has been focused on binary representation, the paradigm can be applied to the different representations discussed in this section.

III. ARITHMETIC: NEW TECHNOLOGIES AND ALTERNATIVE COMPUTING PARADIGMS

As referred to in the previous sections, although arithmetic units and processors have been designed in CMOS-based ASICs and Field Programmable Gate Arrays (FPGAs),
nonconventional number representations are also fundamental for developing computing systems based on new technologies and circuits, such as superconductor devices [79], integrated nanophotonics [94], nanoscale memristor devices and hybrid memories [98], [105], and invertible logic devices [78]. Moreover, new paradigms for designing computers with emergent technologies also cross nonconventional arithmetics to cover a wide range of applications. This section introduces not only those emergent technologies but also two of the most representative new computing paradigms, QC and DNA-based computing [80], [81]. This survey considers both the technology and the new computing paradigms from the perspective of computer arithmetic.

The lowest limit for the energy required to process one bit of information is \( E_b = k_B \times T \times \ln 2 \), where \( k_B \) is the Boltzmann constant and \( T \) the temperature in Kelvin (approximately \( 3 \times 10^{-21} \) Joule for a temperature of 25°C) [82]. However, it has been shown that this energy can be reduced, achieving even near-zero energy dissipation, by using reversible gates [82]. Reversible logic requires not only the reverse mode operation but also a one-to-one mapping between inputs and outputs.

Reversible gates have been implemented in nanotechnologies. Some of the most representative reversible gates are presented in Fig. 12 [81]. The Feynman gate (FG), also known as the Controlled-NOT (CNOT) gate, is a 2 \times 2 reversible gate described by the equations \( P = A \) and \( Q = A \oplus B \), where \( A \) and \( B \) operate as control and data bits, respectively. The Toffoli gate (TG) is a 3 \times 3 reversible logic gate, with \( P = A \), \( Q = B \) and \( R = C \oplus (A \land B) \). A 1-bit Full Adder (FA) can be built with a single Haghparast and Navi gate (HNG) gate by considering \( A, B, C_i \) and \( D = 0 \). The carry out \((C_o)\) bit that results from the addition of \( A, B \) and \( C_i \) is easily computed by (37), since the result of the XOR of three terms is equal to the logic OR whenever the number of terms that assume a value of one is different from two, a situation that never occurs in (37).

\[
C_o = (A \oplus B) \land C \oplus A \land B = A \land C \oplus B \land C \oplus A \land B = A \land C + B \land C + A \land B
\]  

Efficient reversible circuits have been implemented on QCA-based technologies [80]. Binary information is represented by means of Quantum Dots (QDs), semiconductor particles a few nanometers in size having optical and electronic properties that are central to nanotechnology [84]. When illuminated by ultraviolet light, an electron in the QD can be excited to a state of higher energy, while for a semiconducting quantum dot, an electron can transit from the valence band to the conductance band. Reversible QCA-based gates have been applied to design arithmetic units [85], which can be implemented, for example, on DNA molecular structures [86], [87] or on quantum computing devices [88]. Moreover, reversible gates and QCA have been used to implement arithmetic units based on RNS [81], [89].

A. Superconductor logic devices

The AQFP superconductor logic family was proposed as a breakthrough technology towards building energy-efficient computing systems [90]. The dynamic energy dissipation of AQFP logic can be drastically reduced due to the adiabatic switching operations by using AC bias/excitation currents for both the clock signal and power supply.

The circuit in Fig. 13, from [91], represents a typical simple AQFP gate that acts as a buffer. The Quantum-Flux-Parametron (QFP) is a two-junction Superconducting Quantum Interference Device (SQUID). It is based on superconducting loops containing two Josephson junctions \((J_1 \text{ and } J_2)\) and two inductors \((L_1 \text{ and } L_2)\), which are shunted by an inductor \(L_q\) in the center node. An excitation current \(I_{ex}\) is applied, which changes the potential energy of the QFP from a single well to a double well. Consequently, the final state of the QFP, depending only on the direction of the input current \(I_{in}\), is one of two states: the “0” state with an SFQ in the left loop or the “1” state with an SFQ in the right loop (see Fig. 13). A large output current, in the direction defined by the final state of the QFP, is generated in \(L_q\).

One major advantage of AQFP is the energy efficiency potential. For example, experimental results show that the energy dissipation per junction of an 8-bit AQFP-based Carry Look-ahead Adder (CLA) is only \( 24k_BT \times 10^{-21} \) Joule for a temperature of 25°C, being a correct operation demonstrated at a 1-GHz clock frequency. The robustness of AQFP technology has been demonstrated against circuit parameter variations, as well as the potential towards implementing very large-scale integrated circuits using AQFP devices. A more systematic and automatic synthesis framework, as well as synthesis results on a larger number of benchmark circuits, is presented in [90]. The automatic synthesis on 18 benchmark circuits, including 11 circuits from the ISCAS-85 benchmark suite and a 32-bit RISC-V Arithmetic Logic Unit (ALU), is performed using a standard cell library of AQFP technology, with 4-phase clock signals.
The results compare an AQFP 10kA/cm² standard cell library with a TSMC 12 nm FinFET one. The results show the consistent energy benefit of the AQFP technology, which is able to achieve an up to 10-fold improvement in energy consumption per clock cycle in comparison to the 12 nm TSMC technology.

Recent research has highlighted the interest in nonconventional arithmetic also for AQFP technology. In addition to the ultra-high energy efficiency, it is pointed out that this technology has two characteristics that make it especially suitable for implementing SC [79]. One is its deep pipelining nature, since each AQFP logic gate is connected to an AC clock signal, thus requiring a clock phase, which makes it difficult to avoid Read after Write (RAW) hazards in conventional binary computing. The other is the opportunity to obtain a true Random Number Generation (RNG) using single AQFP buffers.

**B. Integrated nanophotonics**

Integrated nanophotonics is another emergent technology that takes advantage of nonconventional arithmetic, in particular RNS [93], [94]. The inherent parallelism of RNS and the energy efficiency of integrated photonics can be explored to build high-speed RNS-based arithmetic units supported on integrated photonics switches. RNS arithmetic is performed through spatial routing optical signals in $2 \times 2$ Hybrid Photonic-Plasmonic (HPP) switches, following a computing-in-the-network paradigm. The block diagram of these $2 \times 2$ HPP switches is represented in Fig. 14. It is fabricated by using indium tin oxide (ITO) as the active index modulation material [95]. In the bar state, the light will propagate straight into the bar arm (Fig. 14a), while in the cross state, lights are routed to the opposite (cross) arm output (Fig. 14b). A voltage signal is applied to the switch to control the guidance of the input light to the desired output port. Theoretically, an HPP switch could be operated at 400 GHz. However, the speed of the whole system is limited by the speed of the modulators and photodetectors, as well as the electronic circuit used to control the routing [94].

As discussed before, RNS arithmetic is applied digitwise; thus, independently for each digit, a number of those switches are interconnected to perform the computation. This is necessary to make systems scalable, since the number of switches grows with the square of the number of bits. For example, for modular addition, the shifting RNS adder explores the shifting property: a modulo $m$ adder requires a set of diagonally aligned $m-1$ HPP switches per each of the $(m-1)$ levels, with $(m-1)^2$ switches in total. For example, Fig. 15a illustrates an adder modulo $m = 5$, requiring $4^2$ switches organized in 4 levels. When computing $x + y$, $x$ can be represented by the input light source, while $y$ specifies electrically the number of levels in which the switches are in the cross state (in the example, levels 1 to 3 for adding 3). For this shift-level selection, TC is used to enable level switching in the RNS channels, as presented in Section II-B.

The ASD RNS adder uses a routing network as the computing device [93]. To compute $x + y$, $x$ is represented by the input light source, and $y$, by the switch states defined by binary electrical signals. By storing the precalculated states that satisfy the all-to-all nonblocking routing network in an Look up Table (LUT), the input light is routed to the resulting output port. Fig. 15b depicts a design for $m = 5$ and the light path used to calculate, as in the previous example, $(4 + 3)_5 = 2$, now through an ASD network. A modulo $m$ ASD RNS adder requires $(m-2)^2/2 + 2$ switches (for odd $m$). If the number of switches required by the ASD RNS adder is less than that for the shifting RNS adder, then more complex electronic control circuitry, typically including LUTs, is required.

Integrated nanophotonic RNS-based arithmetic processors exhibit a very short computational execution time, achieved by optical propagation through an integrated nanophotonic router, on the order of dozens of picoseconds. In comparison to All-Optical-Switch (AOS) technology, also used for RNS-based arithmetic [96], integrated nanophotonics requires a relatively small area due to the compact transistor size. The energy per operation, on the order of fJ/bit, has been shown to be orders of magnitude lower [93].

**C. Nanoscale memristors and hybrid memories**

Several alternatives to existing semiconductor memories, supported by advances in technologies and devices, have recently been proposed [97]. This has led, for example, to the re-emergence of PIM architectures based on Non-Volatile Memory (NVM) and 3D monolithic technologies to meet the demands of extensive data processing for current and next-generation computing. This subsection discusses how nonconventional arithmetic plays an important role in memory technologies, namely, to perform PIM.

Hybrid memories combine CMOS with non-CMOS devices in a single chip as one alternative to existing semiconductor memories [98]. These memories use nanowires, carbon
nanotubes, molecules and other technologies to design the memory cell arrays, while CMOS devices are used to form the peripheral circuitry. One of the first articles to investigate the application of RRNS to improve the reliability of hybrid memories was published in 2011 [98]. A new RRNS code is proposed for a 6-moduli set, where two are nonredundant moduli and four are redundant moduli and maximum likelihood decoding is applied to resolve the ambiguity that occurs for less than 10% of the decoded data.

The CREEPY core, referred to in Section II-B also adopts RRNS to achieve computational error correction for new devices with signal energies approaching the $k_B T$ noise floor [25], such as the Ferroelectric Transistors (FEFET) [99]. The 4T nonvolatile differential memory based on FEFETs is a good energy-efficient candidate for PIM [100].

Memristors, a contraction of the terms ‘memory’ and ‘resistor’, are devices that behave similar to a nonlinear resistor with memory [101]. Memristors can be used to combine data storage and computation in memory, thus enabling non-Neumann PIM architectures [102]. CNNs have been fully hardware-implemented using memristor crossbars, which are cross-point arrays with a memristor device at each intersection [103]. The processing-in-memory-and-storage (PIM) project also adopts the RRNS for memristor-based technologies [104]. Moreover, it has been shown that SC is adequate to perform PIM on nanoscale memristor crossbars, allowing the extension of the flow-based crossbar computing approach to approximate stochastic computing [105]. Applications of in-memory stochastic computing to a gradient descent solver and a k-means clustering processor can be found in [57].

An end-to-end brain-inspired HDC nanosystem using the heterogeneous integration of multiple emerging nanotechnologies was proposed in [106]. It uses monolithic 3D integration of Carbon Nanotube Field-Effect Transistors (CNFETs) and Resistive Random-Access Memory (RRAM). Due to their low fabrication temperature, (1,952) CNFETs and (224) RRAM cells are integrated with fine-grained and dense vertical connections between computation and storage layers. Integrating RRAM and CNFETs allows creating area- and energy-efficient circuits for HDC.

Multi-level cell (MLC) is used to store multiple bits in a single RRAM cell to reduce the hardware overhead. However, the accuracy of ANNs deteriorates more due to resistance variations, which is quite relevant in binary coding. The error-tolerance feature of SC can also be explored to mitigate the reliability problem of RRAM MLC technology [107].

D. Invertible logic circuits

Invertible logic provides a reverse mode for which the output is fixed and the inputs take on values consistent with the output [78]. It is less restrictive than reversible logic. For example, the two-input OR logic gate is invertible if when the output is kept at $c = 1$, the inputs $(a, b)$ alternate equally, with a probability of 33%, between the values $(1, 1), (0, 1)$, and $(1, 0)$, but it is not reversible.

Boltzmann machine structures [83] are underlying structures that have been proposed to design invertible logic circuits [78]. These machines can be represented as networks or graphs of simple processing elements interconnected (e.g., Fig. 16a for a two-input gate $Y = A \land B$) with signals taking the binary values $+1$ or $1$. Every node in the undirected graph is fully connected to all others through bidirectional links, and an individual bias value is assigned to each node. The binary state and $i^{th}$ node output $(m_i)$ are computed by (38): after calculating the weighted sum of all input connections (matrix $J$) and adding the bias terms $(h)$, the nonlinear activation function ($\tanh$) is applied. A noise source $\text{rnd}(-1,+1)$ (uniformly distributed random real number between $-1$ and $+1$) is introduced to avoid getting trapped in local minima, and finally, the sign function ($\text{sgn}$) provides the binary outputs $+1$ or $-1$. $I_0$ is a scaling factor (an inverse pseudo-temperature). States are categorized as valid and invalid states. If the nodes are unconstrained, the system fluctuates among all possible valid states. A logic value of ‘0’ is assigned to $m_i(t) = -1$, and ‘1’ is assigned to $m_i(t) = 1$.

$$m_i(t + \Delta t) = \text{sgn}(\text{rnd}(-1,+1) + \tanh(I_i(t + \Delta t)))$$

$$I_i(t + \Delta t) = I_0(h_i + \sum_j{J_{ij}m_j(t)})$$

(38)

The values of $h$ and $J$ for the AND gate are shown in Fig. 16b. The $h$ vector elements correspond to the bias weights close to the node in the graph (in the order $h_A$, $h_B$, $h_1$, $h_2$ for $A$, $B$, and $Y$ for $Y$) (16a). Matrix $J$, which, as expected, is symmetric, represents the weight of the bidirectional connections, following also the order $A, B, Y$. The gate is invertible; thus, for example, when $Y$ is clamped to 1, the only valid state is $(A = 1, B = 1)$.

Fig. 16: Boltzmann Machine structure for the AND gate [78]: three nodes, those denoted $A$ and $B$ refer to binary inputs, and $Y$ the output

The organization of the nodes, which operate in parallel, and their type, simple processing elements with binary inputs and outputs, make Boltzmann machines well suited for the SC nonconventional arithmetic discussed in the previous section [78]. In Fig. 17, an FSM state, as shown in Fig. 10, is stored as the accumulator: for the highest state of the FSM ($S_{N-1}$), a positive input will not cause any increase in the value stored in the accumulator; the inverse also holds true for negative input signals and the lowest state. The scaling values in (38) are included in the weight $w_i$ terms in Fig. 10.

SC can be used to implement arithmetic functions based on Boltzmann machines. A graph of the Boltzmann machine for an invertible 1-bit FA is represented in Fig. 18.
processing. Following arithmetic operations to be performed through parallel support addresses and operations on single strands, thus not allowing iterative and parallel processing to be carried out. Moreover, earlier DNA computing models do not allow the iterative application of operations to implement arithmetic operations, overcoming the limitations of the previous models and allowing iterative and parallel processing to be carried out.

The DNA found in living cells is composed of four bases: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). Each base is attached to its neighbor base in the sequence via hydrogen bonds. The DNA double helix is composed of two complementary single strands that bond with each other.

E. DNA-based computing

The DNA found in living cells is composed of four bases: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). Each base is attached to its neighbor base in the sequence via hydrogen bonds. The DNA double helix is composed of two complementary single strands that bond with each other.

Fig. 17: Simplified processing element using SC [78]: \( w_i \) represents the factor associated to the \( I_i \) inputs; \( w_{\text{rnd}} \) represents the weight associated to the noise source.

Fig. 18: Boltzmann Machine structure for the 1-bit FA [78]: five nodes, those denoted \( A, B, C_i \) refer to binary inputs, and \( S \) and \( C_o \) the outputs.

\( B \) and \( C_i \) are the inputs, and \( C_o \) and \( S \) are the outputs of the 1-bit FA [78]. In this case, the \( h \) vector elements in Fig. 18b are all zero. The symmetric matrix \( J \) follows the order \( A, B, C_i, S, C_o \). Invertible multipliers can also be implemented, which can be used for factorization. A 5-by-5-bit optimized invertible multiplier/divider/factorizer occupies 53,818 \( \mu \text{m}^2 \), which is 13 times the area of the conventional binary logic counterparts [78]. For the latency, the convergence cycles are different depending on the outputs but are orders of magnitude faster than the alternatives for invertible operations.

Test tubes contain sets of strands representing binary numbers, either the operands or the results of arithmetic operations. The functional formulation of the operations on the Sticker-based DNA model allows the setup of algorithms for performing DNA-based computing [111]. Three main operations are essentially considered to implement logic and arithmetic operations: Combine, Separate and Set.

- Combine(\( T_d, T_s_1, \cdots, T_s_n \)) - pour the contents of tubes \( T_{s_i} \) into \( T_d \) and then empty \( T_{s_i} \); formally, using the set theory, \( T_d \equiv \bigcup_{i} T_{s_i} \cap T_{s_i} \equiv \emptyset \).
- Separate(\( T_s, i, B_{[1]}, B_{[0]} \)) - Separate the contents of \( T_s \) based on the value of the \( i \)th bit; the DNA strands with an \( i \)th bit equal to “1” are inserted into tube \( B_{[1]} \), while the other DNA strands, into tube \( B_{[0]} \).
- Set(\( T_{s_d, i} \)) - Set the \( i \)th bit of all DNA strands in tube \( T_{s_d} \), which serve as the source and destination in this operation.

Several logic and arithmetic operations on the Sticker-based DNA model have been proposed [111]: the bitwise AND operation over two \( n \) bit vectors is presented in Algorithm [1] and the \( n \)-bit integer ADD, in Algorithm [2]. In Algorithm [1], the output of the AND operation over the operands represented by the strands in source tubes \( T_{s_1} \) and \( T_{s_2} \) is made available in the destination tube \( T_d \). After pouring the contents of tubes \( T_{s_1} \) and \( T_{s_2} \) into the auxiliary tube \( T_a \) (line [1]), the strands in this tube are separated according to the value of the bits in each of the \( n \) positions in tubes \( B_{[1]} \) and \( B_{[0]} \) (line [3]): whenever the value of the \( i \)th bit in the two strands is “1”, the \( i \)th bit of the DNA strand in tube \( T_d \) is set (line [3]). Strands are re-combined in tube \( T_a \) in line [7].

Fig. 19: Memory strands, stickers and DNA-based number representation.

More recently, models such as the Sticker-based DNA model have been proposed [110]. The sticker-based DNA model is used to represent fixed-point and floating-point numbers, overcoming the limitations of the previous models and allowing iterative and parallel processing to be carried out [111]. With the DNA sticker, a binary number is represented through two groups of single-stranded DNA molecules: \( i \) the memory strand, a long DNA molecule that is subdivided into nonoverlapping segments and \( ii \) a set of stickers, i.e., short DNA molecules, each with the length of a segment of the memory strand. Each one of the nonoverlapping segments represents a bit, and a sticker is complementary to one and only one of the nonoverlapping segments. As depicted in Fig. 19 a segment on the memory strand annealed to the matching sticker represents a “1”; otherwise, the value of the bit is “0”. Fig. 19 a segment on the memory strand annealed to the matching sticker represents a “1”; otherwise, the value of the bit is “0”.

Several logic and arithmetic operations on the Sticker-based DNA model have been proposed [111]: the bitwise AND operation over two \( n \) bit vectors is presented in Algorithm [1] and the \( n \)-bit integer ADD, in Algorithm [2]. In Algorithm [1], the output of the AND operation over the operands represented by the strands in source tubes \( T_{s_1} \) and \( T_{s_2} \) is made available in the destination tube \( T_d \). After pouring the contents of tubes \( T_{s_1} \) and \( T_{s_2} \) into the auxiliary tube \( T_a \) (line [1]), the strands in this tube are separated according to the value of the bits in each of the \( n \) positions in tubes \( B_{[1]} \) and \( B_{[0]} \) (line [3]): whenever the value of the \( i \)th bit in the two strands is “1”, the \( i \)th bit of the DNA strand in tube \( T_d \) is set (line [3]). Strands are re-combined in tube \( T_a \) in line [7].

Strands are re-combined in tube \( T_a \) in line [7].
Algorithm 1 AND($T_{s1}, T_{s2}, n$:in; $T_d$:out)

Require: A blank strand of $n$ bits ($0 \ldots 0$) into $T_d$
Ensure: bit stream in $T_d$ = bit stream in $T_{s1} \land$ bit stream in $T_{s2}$
1: Combine($T_a, T_{s1}, T_{s2}$) ($T_a$: auxiliary Tube)
2: for all bit $0 \leq i < n$ do
3: Separate($T_a, i, B_{[1]}, B_{[0]}$)
4: if $B_{[0]}$ is empty then
5: Set($T_a, i$)
6: end if
7: Combine($T_a, B_{[1]}, B_{[0]}$)
8: end for

Algorithm 2 ADD($T_{nc}, T_{s1}, T_{s2}, n$:in; $T_d$:out)

Require: A blank strand of $n$ bits ($0 \ldots 0$) into $T_d$
Ensure: bit stream in $T_d$ = bit stream in $T_{s1} \oplus$ bit stream in $T_{s2}$
1: Combine($T_{nc}, T_{s1}, T_{s2}$) ($T_{nc}$: No-carry auxiliary Tube)
2: for bits $i = 0$ to $n - 1$ do
3: if $T_{nc}$ is not empty then
4: Separate($T_{nc}, i, B_{[1]}, B_{[0]}$)
5: if neither $B_{[0]}$ nor $B_{[1]}$ is empty then
6: Set($T_d, i$) {bits in position $i$ have different values}
7: Combine($T_{nc}, T_{s1}, T_{s2}$)
8: else
9: if $B_{[0]}$ is empty then
10: Combine($T_c, T_{s1}, T_{s2}$) ($T_c$: (with-)carry auxiliary Tube)
11: else
12: Combine($T_{nc}, T_{s1}, T_{s2}$) {bits in position $i$ are both $0$}
13: end if
14: end if
15: else
16: Separate($T_c, i, B_{[1]}, B_{[0]}$)
17: if neither $B_{[0]}$ nor $B_{[1]}$ is empty then
18: Combine($T_{nc}, T_{s1}, T_{s2}$) {bits in position $i$ have different values}
19: else
20: if $B_{[0]}$ is empty then
21: Set($T_d, i$) {bits in position $i$ are both $1$}
22: Combine($T_c, T_{s1}, T_{s2}$)
23: else
24: Set($T_d, i$) {bits in position $i$ both $0$}
25: Combine($T_{nc}, T_{s1}, T_{s2}$) {bits in position $i$ are both $0$}
26: end if
27: end if
28: end if
29: end for

Algorithm 2 applies the ADD operation over the $n$-bit operands represented by strands in source tubes $T_{s1}$ and $T_{s2}$, and the sum is deposited into the destination tube $T_d$. Although the addition algorithm is supported by binary arithmetic, we present it to illustrate how arithmetic is implemented in biological substrates. The auxiliary tubes $T_{nc}$ and $T_c$ are used to register the absence or existence of the carry bit, respectively. By pouring the contents of tubes $T_{s1}$ and $T_{s2}$ into the auxiliary tube $T_{nc}$, it is considered that the initial carry-in is equal to “0” (line 1). For the Least Significant Bit (LSB), strands in the $T_{nc}$ tube (line 2) are separated according to the value of this bit in tubes $B_{[1]}$ and $B_{[0]}$ (line 3). If neither $B_{[1]}$ and $B_{[0]}$ are empty, this means that the LSBs have different values (line 4). Since carry-in is not considered, the LSB of the DNA strand in tube $T_d$ is set (line 5). No carry-out is generated; thus, both strands are placed back in the $T_{nc}$ tube (line 7). On the other hand, if both LSBs take the value 1 (line 8), the sum bit is “0”, and we only have to put both strands in the $T_c$ (line 10) to signal that a carry-in exists for the next bit. For the last condition in line 11, both LSBs are “0”, and we only have to return both strands to $T_{nc}$ (line 12). The second half of Algorithm 2 from line 13 is similar to the first but for the case where a carry-in exists, which means tube $T_{nc}$ is empty and tube $T_c$ contains the input strands. All this processing is iteratively repeated from the LSB to the Most Significant Bit (MSB) (line 2).

Current DNA computational systems are constrained by their poor integration capability, complex device structures or limited computational functions. However, the ability to build efficient DNA logic gates and cascade circuits based on polymerase-mediated strand displacement synthesis has been shown [112]. The pillars are a single-rail AND gate built with three strands and a single-rail OR gate built with two strands. Dual-rail DNA logic gates were built by parallelizing a single-rail AND gate and a single-rail OR gate to construct a logical expression. The NOT gate, fundamental to the construction of general logic systems, is difficult to build but can be achieved by reversing the definition of the strands and constructing a single-rail AND gate and a single-rail OR gate [112].

A DNA ALU, with four 1-bit functions (FA, AND, OR, and NAND) and the decoding and controlling logic, was constructed [112]. It was built with 16 equivalent logic gates and consists of 27 DNA species and 74 DNA strands. After purifying the strands with PolyAcrylamide Gel Electrophoresis (PAGE), the logic gates and circuits are tested, and the results are visualized with the real-time Polymerase Chain Reaction (PCR); PAGE can also be followed by gel imaging. The results in [112] show that it is possible to integrate components to implement DNA computer systems. Leakage is the main challenge, especially when the size scales up. The limited purity of commercial chemosynthetic strands and DNA components is the primary source of leakage. Moreover, although any kind of DNA strand can be synthesized using biological methods, the preparative step requires a long time in practice, as well as the preparation of inputs [109].

Although the DNA manipulation required in the models has already been realized in the laboratory and the procedures have been implemented in practice, some defects exist in the procedures, thus hindering practical use. This is an example of the application of nonconventional arithmetic to DNA-based computing. The RRNS has been applied to overcome the negative effects caused by the defects and instability of the biochemical reactions and errors in hybridizations for DNA computing [113]. Applying the RRNS 3-moduli set $\{2^n - 1, 2^n + 1, 2^{n+1}\}$ to the DNA model in [114], as discussed in Section II-B, leads to one-digit error detection. The parallel RRNS-based DNA arithmetic improves the reliability of DNA computing while at the same time simplifies the DNA encoding scheme [113].
the gates are represented by Since operations on a qubit preserve the norm of the vectors, algebraic properties, such that H, responds to α Regarding measurement, the superposition state of a quantum system. A qubit, represented by the vector α the basis vectors with coefficients such as

\[ |x\rangle = \alpha |0\rangle + \beta |1\rangle ; |\alpha|^2 + |\beta|^2 = 1. \] (39)

Regarding measurement, the superposition \( \alpha |0\rangle + \beta |1\rangle \) corresponds to \( |0\rangle \) with probability \( |\alpha|^2 \) and \( |1\rangle \) with probability \( |\beta|^2 \).

Common simple qubit gates are represented in Fig. 20. Since operations on a qubit preserve the norm of the vectors, the gates are represented by \( 2 \times 2 \) unitary matrices. Some algebraic properties, such that \( H = (X + Z)\sqrt{2} \) and \( S = T^2 \), are useful to correlate some of these quantum gates.

A two-qubit system can be represented by a vector in the Hilbert space \( \mathcal{H}^2 \otimes \mathcal{H}^2 \), with \( \otimes \) denoting the tensor product, which is isomorphic to \( \mathbb{C}^4 \). Thus, the basis of \( \mathcal{H}^2 \otimes \mathcal{H}^2 \) can be written as:

\[ |0\rangle \otimes |0\rangle, |0\rangle \otimes |1\rangle, |1\rangle \otimes |0\rangle, |1\rangle \otimes |1\rangle \]

and \( |a\rangle \otimes |b\rangle \) is often expressed as \( |a\rangle |b\rangle \) or \( |ab\rangle \). Generally, the state of an \( n \)-qubit system is expressed by \( |\psi\rangle \) in \( \mathcal{H}^2 \).

\[ Y = \sum_{b \in \{0,1\}^n} c_b |b\rangle \quad \text{with} \quad \sum_{b} |c_b|^2 = 1 \quad (40) \]

with the state of an \( n \)-particle system being represented in a \( 2^n \)-dimensional space. The exponentially large dimensionality of this space makes quantum computers much more powerful than classical analogue computers, the state of which is described only by a number of parameters proportional to the size of the system. By contrast, \( 2^n \) complex numbers are required to keep track of the state of an \( n \)-qubit system.

If the qubits are allowed to interact, then the closed system includes both qubits together, meaning that the qubits are entangled. When two or more qubits are entangled, there exists a special connection between them; the outcome of the measurement of one qubit is correlated to the measurement of the other qubits. The quantum version of the CNOT gate in Fig. 12 is an example of the interaction between 2 qubits that operate on the same basis:

\[ |0\rangle \rightarrow |00\rangle \quad |01\rangle \rightarrow |01\rangle \quad |10\rangle \rightarrow |11\rangle \quad |11\rangle \rightarrow |10\rangle \]

The CNOT gate flips the state of the target (t) qubit according to the state of the first control (c) qubit. If the control qubit is set to \( |1\rangle \), then the target qubit is flipped; otherwise, nothing is done with the target qubit, which action can be represented as: \( |c\rangle |t\rangle \rightarrow |c\rangle |c \oplus t\rangle \).

Fig. 21 provides the CNOT gate matrix and circuit representations.

A quantum circuit that implements a 1-bit FA, acting on the superposition of states, is presented in Fig. 22. With the application of the equations of the classic reversible logic (Fig. 12) to the circuit in Fig. 22 the output equations for the full adder are achieved: \( P = A, Q = B, S = A \oplus B \oplus C, \) and \( C_0 = AB + AC + BC \). Thus, if, for example, the four qubits of the input are set to the state \( |ABCX\rangle = |1010\rangle \), then the system goes, with a probability equal 1, to a state that provides an output \( |PQSC_0\rangle = |1001\rangle \).

Quantum gates and quantum computers have already been developed, for example, TGs based on quantum mechanics were successfully realized more than ten years ago at the
In the next section, we refer the capacity of QC to attack the current security measures of cryptographic protocols.

IV. APPLICATIONS IN EMERGING AREAS

Postquantum cryptography and machine learning, in particular DL, have been selected as case studies of applications that use nonconventional arithmetic supported by the technologies and systems presented in the previous sections. The criterion for selection was based on the importance of these applications in emerging areas, particularly those that can benefit the most from nonconventional arithmetic.

A. Deep learning

A type of DL dominant in a broad spectrum of applications is the CNN [120], a class of the general Deep Neural Network (DNN). The CNN encompasses two main different phases of operation: training and inference. The training is an offline process, usually performed in powerful servers, for learning features from (typically massive) databases. The inference applies the trained model to predict and classify according to the features “learned” by the neural network. A CNN architecture consists of layers of convolution, activation, pooling, and fully connected neural networks [120]. The convolution kernels, represented by the weight $w$ in the general equation (41), are applied to identify features by also adding a scalar bias $b$ to (41). For the activation, nonlinear functions, such as the sigmoid or the Rectified Linear Unit (ReLU) in (42), are applied to obtain the values of the feature maps. The pooled layer maps the features into a lower resolution space, for example, with the function in (43). Fully connected neural networks compute over reduced spaces, applying weights to the inputs of artificial neurons that “fire” based on nonlinear functions. The large amount of computation and the high memory requirements are challenges faced in the development of efficient CNNs, especially in devices with autonomy and resource constraints. It will be shown how RNS, LNS, associated with fixed-point and FP representations, and SC can be used to face these challenges.

$$\text{conv} \rightarrow \sum_i w_i y_i + b \quad (41)$$
$$\text{act} \rightarrow h_t = \tanh(y); h_{\text{ReLU}} = \max(0, y) \quad (42)$$
$$\text{pool} \rightarrow p_{\max} = \max(y); p_{\text{ReLU}} = \sqrt{\sum y_i^2} \quad (43)$$

Although several proposals have been made to apply RNS to compute a CNN, only a few can be fully computed in the RNS domain [121]–[123]. These complete solutions mitigate the overhead imposed by the nonlinear functions of the activation and pooling layers by carefully selecting the RNS moduli sets. In [121], the 4-tuple $\{2^n \pm 1, 2^{n+1} \pm 1\}$ takes advantage of the fact that $M$ in (9) is odd. Thus, (41) can be applied to perform the comparison required to compute the $\max$ function in (42) and (43). The fully RNS-based CNN accelerator in [123], i.e., Res-DNN, is supported by the Eyeriss architecture [124] and the RNS arithmetic for the 3-moduli set $\{2^n - 1, 2^n, 2^{n+1} - 1\}$, preventing overflow by using the BE algorithm referenced in [II-B](extending $2^n$ to the modulo.

Fig. 22: Quantum 1-bit FA: Diagram of a circuit using CNOT and TG reversible gates– behaviour simulated on the Quantum Inspire (QI) platform [168].

Fig. 23: The transmon qubit and the quantum chip [169]: a type of superconducting charge qubit; the superconductors are capacitatively shunted in order to decrease the sensitivity to charge noise.

University of Innsbruck [115]. The IBM Q systems are physically supported by the transmon qubit represented in Fig. 23. The transmon qubit circuit corresponds to Josephson junctions, kept at a very low temperature, shunted by an additional large capacitance (Fig. 1 [116]). Although the transmon qubit is closely related to the Cooper PairBox (CPB) qubit [117], it is operated at a significantly different ratio of Josephson energy for the charging energy. It overcomes the main weakness of the CPB by featuring an exponential gain in the insensitivity to charge noise [116]. IBM Q processors are composed of transmon qubits that are coupled and addressed through microwave resonators, as depicted in Fig. 23. Several quantum computers have been constructed based on that technology, the most powerful and recent one being a 53-qubit system [118].

In the next section, applications of the nonconventional arithmetic, new technologies and systems to emerging applications are discussed. While QC algorithms have been derived to solve linear systems of equations with low complexity [119], in the next section, we refer the capacity of QC to attack the
2^{n+r})$. A dynamic range partitioning approach is adopted to implement the comparison operation in the RNS domain [125], while for sign detection in the ReLU unit, the MRC method is adopted [126].

Apart from the large amount of computing, CNNs also require high memory bandwidth due to the large volume of data that must be moved between the memory and processing units. CNNs fully computed in the RNS domain are also proposed for PIM (RNSnet) [122]. RNSnet simplifies the fundamental CNN operations and maps them with the support of RNS to in-memory addition and data access. The nonlinear activation functions are approximated using a few of the first terms of the Taylor expansion, and the comparison for pooling is achieved by simply aggregating streams to compute $\log_{2}(\text{conv})$ for the activation function is realized through an FSM, and decrement. To improve the signal-to-noise ratio, weight normalization and upscaling are performed, while the overhead of binary-to-stochastic conversion is reduced by sharing number generators. In [131], the linear rectifier used for the activation function is realized through an FSM, and pooling is achieved by simply aggregating streams to compute either the average or the max function. Compared with the existing CNN deterministic architectures, the stochastic-based architecture can achieve, at the cost of a slight degradation in computing accuracy, significantly higher energy efficiency at low cost. The SC-DCNN is another example of nonconventional arithmetic applied to design CNNs [130], [131], [133]–[137]. In [130], the polling layers are implemented with an improved version of the SC max unit based on the FSMs presented in Section II-C [132]. The SC-based ReLU, SReLU, is also implemented through an FSM that mimics the three key operations of the artificial neuron: integration, output generation, and decrement. To improve the signal-to-noise ratio, weight normalization and upscaling are performed, while the overhead of binary-to-stochastic conversion is reduced by sharing number generators. In (44), the accuracy of the multiply and add operations with \(x < 1\) can be rewritten as (45). (45) can be easily generalized for any number of terms, \(i\) and \(\tilde{p}_i\), leading to the complete computation in the logarithmic domain of the convolution layers and the remaining layers of the CNN [12]. Moreover, an end-to-end training procedure based on the logarithmic representation is proposed in [12].

\[
\log_{2}(\text{conv}) = \log_{2}(2^{\tilde{p}_1} + 2^{\tilde{p}_2}) = \log_{2}[2^{\tilde{p}_1}(1 + 2^{\tilde{p}_2})] = \tilde{p}_1 + \log(1 + 2^{\tilde{p}_2} / 2^{\tilde{p}_1}) \approx \tilde{p}_1 + \text{BitShift}(1, \tilde{p}_2 - \tilde{p}_1)
\]

In (45), the accumulation is still performed in the linear domain, with \(\text{BitShift}(1, z)\) denoting the function that bit-shifts 1 by an integer \(z\) in the fixed-point arithmetic. To reach the method in Fig. 24 the accumulation is also performed in the log-domain by using the approximation \(\log_{2}(1 + x) \approx x\) for \(0 \leq x < 1\). By considering the accumulation of only two terms, i.e., \(\tilde{p}_1 = \tilde{x}_1 + \tilde{w}_1\) and \(\tilde{p}_2 = \tilde{x}_2 + \tilde{w}_2\), (44) can be rewritten as (45). (45) can be easily generalized for any number of terms, \(i\) and \(\tilde{p}_i\), leading to the complete computation in the logarithmic domain of the convolution layers and the remaining layers of the CNN [12]. Moreover, an end-to-end training procedure based on the logarithmic representation is proposed in [12].

SC is another example of nonconventional arithmetic applied to design CNNs [130], [131], [133]–[137]. In [130], the polling layers are implemented with an improved version of the SC max unit based on the FSMs presented in Section II-C [132]. The SC-based ReLU, SReLU, is also implemented through an FSM that mimics the three key operations of the artificial neuron: integration, output generation, and decrement. To improve the signal-to-noise ratio, weight normalization and upscaling are performed, while the overhead of binary-to-stochastic conversion is reduced by sharing number generators. In (44), the accuracy of the multiply and add operations with \(x < 1\) can be rewritten as (45). (45) can be easily generalized for any number of terms, \(i\) and \(\tilde{p}_i\), leading to the complete computation in the logarithmic domain of the convolution layers and the remaining layers of the CNN [12]. Moreover, an end-to-end training procedure based on the logarithmic representation is proposed in [12].

\[
\log_{2}(\text{conv}) = \log_{2}(2^{\tilde{p}_1} + 2^{\tilde{p}_2}) = \log_{2}[2^{\tilde{p}_1}(1 + 2^{\tilde{p}_2})] = \tilde{p}_1 + \log(1 + 2^{\tilde{p}_2} / 2^{\tilde{p}_1}) \approx \tilde{p}_1 + \text{BitShift}(1, \tilde{p}_2 - \tilde{p}_1)
\]

Fig. 24 shows the samples and the weights for convolution represented in the log-domain [12] as \(\tilde{x}_i = \text{Quantize}(\log_{2}(x_i))\) (3-bit width) and \(\tilde{w}_i = \text{Quantize}(\log_{2}(w_i))\) (4-bit width), respectively.
the result of the first multiplication and performing the subsequent multiplications with the differences of the successive weights [137].

An SC-based DL framework was implemented with AQFP technology [79], a superconducting logic family that achieves high energy efficiency, as described in Section III-A. Through an ultraefficient stochastic number generator and a high-accuracy subsampling (pooling) block, both in AQFP, the implemented CNN achieves a four-orders-of-magnitude higher energy efficiency than the CMOS-based implementation while maintaining a 96% accuracy for the MNIST dataset.

B. Cryptography

Difficult mathematical problems are used to ensure the security of public-key cryptographic systems. These systems are designed so that the derivation of the private key from the public key is as difficult as computing the solution to problems such as the factorization of large integers (e.g., RSA [138]) or the discrete logarithm (e.g., Elliptic Curve (EC) cryptography [170]). The number of steps required to solve these problems is a function of the length of the input; thus, the dimension of keys is enlarged to make the systems secure (e.g., private keys larger than 512 and 1024 bits).

Modular reduction (modulo \( N \)) is a fundamental arithmetic operation in public cryptography for mapping results back to the set of representatives \( \{0, \ldots, N - 1\} \). The Montgomery algorithm for modular multiplication replaces the costly reduction for a general value \( N \) with a reduction for \( L \), typically a power of 2 [139]. However, since in RNS, the reduction modulo power of 2 is inefficient, an RNS variation of the Montgomery modular multiplication algorithm relies on the BE method presented in [47] by choosing \( L = M_1 \) and \( \text{GCD}(M_1, M_2) = 1 \) [140], [141]. Similarly, the reduction modulo \( L \) corresponds to the reduction modulo of the dynamic range \( M_1 \), which automatically relies on the modulo operation in each one of the RNS channels of \( S_1 \). Moreover, \( (N)_{M_1} \) and \( (M_1)_{M_2} \) exist, which are required for the precomputed constants used in the modified Montgomery modular multiplication algorithm [142].

A survey on the usage of RNS to support these classic public-key algorithms can be found in [142]. Multimoduli RNS architectures have also been proposed to obfuscate secure information. The random selection of moduli is proposed for each key bit operation, which makes it difficult to obtain the secret key, preventing power analysis while still providing all the benefits of RNS [143].

However, if quantum computers become available, this type of public-key cryptography, based on the factorization of large integers or the discrete logarithm, will no longer be secure [144]. Therefore, there is a need to develop arithmetic techniques for alternate systems, usually designated as postquantum cryptography [145]. Quite recently, another survey was published on the application of RNS and SC to this type of emergent cryptography [147]. We present two representative examples of the application of RNS and SC to the Goldreich-Goldwasser-Halevi (GGH) postquantum cryptography and Fully Homomorphic Encryption (FHE), respectively.

A lattice can be seen as a vector space generated by all linear combinations with integer coefficients of a set \( \mathcal{R} = \{\vec{r}_0, \ldots, \vec{r}_{n-1}\} \), with \( \vec{r}_i \in \mathbb{R}^m \), of linearly independent vectors, as defined in (46), the rank of the lattice is \( n \), and its dimension is \( m \). Two vectors in \( \text{span}(R) \) are congruent if their difference is in \( \mathcal{L}(R) \).

\[
\mathcal{L}(\mathcal{R}) = \left\{ \sum_{i=0}^{n-1} z_i \vec{r}_i : z_i \in \mathbb{Z} \right\}
\]

Each basis is associated with the parallelepiped:

\[
\mathcal{P}(R) = \left\{ \sum_{i=0}^{n-1} w_i \vec{r}_i : w_i \in \left( -\frac{1}{2}, \frac{1}{2} \right) \right\}
\]

For any point \( \vec{y} = \vec{w} + \vec{x} \in \text{span}(R) \), \( \vec{x} \in \mathcal{P}(R) \), the reduction in the \( \vec{y} \) modulo \( \mathcal{P}(R) \) is defined as \( \vec{x} = \vec{y} \mod \mathcal{P}(R) \). The modular reduction has a different meaning for each basis, since they are associated with different parallelepipeds. An example of this is featured in Fig. 25, where the point \( \vec{c} \) is reduced modulo both \( \mathcal{P}(R) \) and \( \mathcal{P}(B) \), producing two different points, which are represented as triangles, while \( \mathcal{L}(R) = \mathcal{L}(B) \).

![Fig. 25: Two basis \( \vec{r}_0, \vec{r}_1 \) and \( \vec{r}_0, \vec{r}_1 \) of the same lattice, along with the corresponding parallelepipeds in red and grey, are represented; \( \vec{c} \) is reduced modulo the two parallelepipeds [147]](image)

Lattice Based Cryptography (LBC) is supported by the Closest Vector Problem (CVP). This problem consists of a given base \( R \in \mathbb{R}^{n \times m} \), and \( \vec{y} \in \mathbb{R}^m \), finding \( \vec{x} \in \mathcal{L}(R) \) such that \( ||\vec{y} - \vec{x}|| = \min_{\vec{z} \in \mathcal{L}(R)} ||\vec{y} - \vec{z}|| \). The private basis is produced as a rotated nearly orthogonal basis, such that Babai’s round-off [146] provides accurate solutions to the CVP. Rose’s cryptosystem uses bases of an Optimal Hermite Normal Form (OHNF) as the public key, a subclass of Hermite Normal Forms (HNFs), where all but the first column are trivial. The decryption algorithm is modified for its implementation with the RNS. The operation \( \lfloor cR^{-1} \rfloor \) \((\lfloor \cdot \rfloor \) denotes rounding to the nearest integer) can be replaced by the approximation \( \tilde{c} \) of \( \lfloor cR^{-1} \rfloor \) through an RNS Montgomery reduction [139], where \( \tilde{c} = (c, 0, \ldots, 0) \) and the scaling by \( \gamma \) enables the
using integer arithmetic as in (48), where $\hat{c}R^{-1}$ is rewritten using integer arithmetic as in [48], where $\hat{R} = R^{-1}d$ is an integer and $d = \det(R)$.

$$[\gamma c R^{-1}] = \frac{\gamma c \hat{R} - \left< \gamma c \hat{R} \right>_d}{d}$$  (48)

It is shown that the usage of RNS enables parallelizing the decryption in Rose’s cryptosystems to significantly speed up its computation in both CPUs and GPUs [147].

Homomorphic encryption allows performing computations directly on ciphertexts, generating encrypted results as if the operations had been performed on the plaintext and then encrypted. FHE provides malleable ciphertexts, such that when two ciphertexts representing the operands, it will be possible to produce a ciphertext encrypting its product or sum. Structured lattices underpin classes of cryptosystems supporting FHE [148]. There is noise associated with the ciphertexts that grow as homomorphic operations are applied; thus, bootstrapping has been proposed, a technique in which ciphertexts are homomorphically decrypted [148]. Modern FHE systems rely on Ring Learning with Errors (RLWE), for which techniques have been proposed that limit the need for bootstrapping [147].

Batching [149] improves the performance of FHE based on the CRT by allowing multiple bits to be encrypted in a single ciphertext so that one can carry out AND and XOR sequences of bits using a single homomorphic multiplication or addition. For example, in an RLWE cryptosystem, binary polynomials are homomorphically processed in a cyclotomic ring. By noticing that certain cyclotomic polynomials factor modulo two onto a set of polynomials with the same degree, one may take advantage of the CRT to associate a plaintext bit with each one of these polynomials. Homomorphic additions and multiplications then add and multiply the bits modulo their respective polynomial, achieving coefficientwise XOR and AND operations. Rotations of these bits may be accomplished with [150].

The operations that arise from FHE are evaluated, and efficient algorithm-hiding systems are designed for applications that take advantage of those operations in [151]. First, numbers can be represented as sequences of bits through stochastic representations. Since most FHE schemes support batching as an acceleration technique for the processing of multiple bits in parallel, one can map stochastic representations to batching slots to efficiently implement multiplications and scaled additions [15]. Unlike traditional Boolean circuits, the amount of bits required for stochastic computing is flexible, allowing for a flexible correspondence between the unencrypted and encrypted domains.

The homomorphic systems based on SC in [151] target the approximation of continuous functions. These functions are first approximated with Bernstein polynomials in a black-box manner. Hence, function development can take place with traditional programming paradigms, providing an automated way to produce FHE circuits. A stochastic sequence of bits is encrypted in a single cryptogram. It was concluded that stochastic representations are more widely applicable, while a fixed-point representation provides better performance [151].

A quantum computer efficiently factors an arbitrary integer [153], which makes the widely used RSA public-key cryptosystem insecure. It is based on reducing this computation to a special case of a mathematical problem known as the Hidden Subgroup Problem (HSP) [154], [155] (HSP is a group theoretic generalization of the problem of periodicity determination). The Abelian quantum hidden subgroup algorithms in [153], [156] are based on the Quantum Fourier Transform (QFT). The QFT, the counterpart to the Discrete Fourier Transform (DFT) in the classical computing model, is the basis of many quantum algorithms [152], [157], [158].

V. CONCLUSIONS AND FUTURE RESEARCH

This survey covers the main types of nonconventional arithmetic from a holistic perspective, highlighting their practical interest. Several different classes of nonconventional arithmetic are reviewed, such as LNS, RNS, SC and HDC, and their usage in various emergent applications, with different features and based on new technologies, is discussed. We show the importance of nonconventional number representation and arithmetic not only to implement fast, reliable, and efficient systems but also to shape the usage of the technology. For example, SC is useful for FHE, enabling the processing of multiple bits in parallel for batching, but is also important to perform approximate computing on nanoscale memristor crossbars and to overcome the deep pipelining nature of AQFP logic devices. Similarly, RNS is suitable for machine learning and cryptographic applications, but it is also quite useful to mitigate the instability of biochemical reactions for DNA-based computing and to reduce the cost of arithmetic systems for integrated nanophotonics technology. Hyperdimensional representation and arithmetic, inspired by brain-like computing, adopt high-dimensional binary vectors to introduce randomness and achieve reliability in HDC. This type of nonconventional number representation offers a strong basis for in-memory computation, being a promising candidate for taking advantage of the physical attributes of nanoscale memristive devices to perform online training, classification and prediction.

One of the main conclusions that can be drawn from this survey is that the investigation of nonconventional arithmetic must take into account all the dimensions of the systems, which includes not only computer arithmetic theory but also technology advances and the demands of emergent applications. For example, the results of further investigations of the randomness characteristics of SC and HDC may also impact applications that might benefit from approximate computations [66]. At the technological level, these classes of arithmetic may benefit from the ease of nanotechnologies in generating the required random numbers [79] while improving the robustness to noise [77]. The efficiency offered by SC and HDC results in important savings both in terms of the integration area and hardware cost, which make these number representations suitable for the development of autonomous cyberphysical [161] and Internet of Things (IoT) processors [73].
Further research on LNS and RNS should target difficult operations in the respective domains, namely, addition and subtraction in the logarithmic domain and division and comparison in the RNS domain. The accuracy and the implementation cost under different technologies are key aspects for the practical usage of this nonconventional arithmetic. In that respect, the development of tools such as the Computing with the Residue Number System (CRNS), which automates the design of systems based on RNS for exploring the increasing parallelism available in computing devices, is also important [162]. The redundancy of the RRNS and the randomness of SC and HDC make them suitable for improving reliability [113] and supporting the heterogeneous integration of multiple emerging nanotechnologies [106]. Moreover, it will be very useful to investigate efficient converters between nonconventional systems that allow them to be used together in different components of heterogeneous systems and for different applications.

New paradigms of computation will also become the focus of research on nonconventional computer arithmetic. The less disruptive, more straightforward path of research is the combination of nonconventional arithmetic with approximate computing. While for SC, this combination is natural, for the other nonpositional number representations, a research effort focused on this purpose is required. For the other types of new paradigms of computation, such as QC, the quantum computer platforms publicly available, i.e., IBM Q [169] and QI from QuTech [168], provide both simulators and real quantum computers at the back end. They are quite useful for performing an investigation of QC and introduce this topic with a practical component in university curricula. They are valuable instruments for simulating new quantum algorithms and circuits for several applications, extending their usage far beyond the domain of quantum mechanics.

ACKNOWLEDGMENTS

This work was partially supported by the European Union Horizon 2020 research and innovation program under grant agreement No. 779391 (FutureTPM) and by national funds under project number UIDB/50021/2020. I would also like to acknowledge the contributions of all MSc and PhD students that I have supervised over the years. They have been a source of inspiration and challenges in the adventure that research on computer arithmetic has been for me.

REFERENCES


A. Molahosseini, A. Asadpoor, H. Khademolhosseini, S. Sheikhfaal, S. Angizi, S. Thirumala and S. Gupta, “Reconfigurable ferroelectric transistorpart...” [rest of the text is not visible in the provided snippet].


Leonel Sousa received a Ph.D. degree in Electrical and Computer Engineering from the Instituto Superior Técnico (IST), Universidade de Lisboa (UL), Lisbon, Portugal, in 1996, where he is currently a Full Professor. He is also a Senior Researcher with the R&D Instituto de Engenharia de Sistemas e Computadores (INESC-ID). His research interests include computer arithmetic, VLSI architectures, parallel computing and signal processing. He has contributed to more than 200 papers in journals and international conferences, for which he has received several awards, including the DASIP13 Best Paper Award, SAMOS11 Stamatis Vassiliadis Best Paper Award, DASIP10 Best Poster Award, and Honorable Mention Award UTL/Santander Totta for the quality of his publications in 2009 and 2016. He has contributed to the organization of several international conferences, typically as program chair and as a general and topic chair, and has given keynotes in some of them. He has edited four special issues of international journals; he is currently the Associate Editor of the IEEE Transactions on Computers, IEEE Access, Springer JRTIP, and Senior Editor of the Journal on Emerging and Selected Topics in Circuits and Systems. He has co-edited the books Circuits and Systems for Security and Privacy (CRC, 2018) and Embedded Systems Design with Special Arithmetic and Number Systems (Springer, 2017). He is a Fellow of the IET, a Distinguished Scientist of the ACM and a Senior Member of the IEEE.