Proc. of International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA 2024)
1-2 February 2024, Victoria-Seychelles

Simple and Linear Fast Adder of Multiple Inputs and Its Implementation in a Compute-In-Memory Architecture

Juan P. Ramírez
Jalisco, México
jramirez@binaryprojx.com;
jramirez@ddots.com;
Personal Page: www.binaryprojx.com

***Abstract*—A Simple and Linear Fast Adder is proposed that supports parallel addition. Its Compute-In-Memory architecture maximizes efficiency and performance of operations ranging from addition to matrix multiplication. The simple and linear topology of the adder allows for scalability past 64-bit architecture. Copies of the adder can be connected in parallel to generate a rectangular grid that performs addition of multiple inputs and scalar multiplication of two inputs. This grid can be scaled to support efficient matrix multiplication.**

***Index Terms*—Applied Mathematics, Circuits and Systems, Microelectronics, VLSI Circuits, Compute-In-Memory, Fast Adder, Matrix Multiplication**

I. Introduction

An innovative Compute-In-Memory implementation of a (Patent Pending) Simple and Linear Fast Adder is proposed. The design is based on a novel set theory, succinctly outlined to provide a foundation for efficient data representation and arithmetic circuit design. The primary focus here is on the practical applications, particularly within the realm of Artificial Intelligence (AI), where the implications of this proposed circuit are transformative. The scalable, linear, and simple topology of the proposed Simple and Linear Fast Adder not only eases the design and manufacturing process but also positions it as a faster and more energy-efficient alternative to conventional adders. These attributes make it particularly well-suited for applications in AI, where heavy loads of matrix multiplication are a common computational requirement.

Furthermore, the Compute-In-Memory architecture adds another layer of innovation, enhancing overall system efficiency. The integration of computation and memory functions in a seamless manner contributes to faster processing speeds and reduced data transfer requirements. Beyond AI, the scalability and efficiency of the proposed circuit have broader applications in areas that heavily rely on matrix multiplication, such as signal processing, image and video processing, and scientific simulations. The simplicity of the circuit's topology not only facilitates cost-effective design and manufacturing but also opens doors for wider adoption across various industries.

In summary, a Compute-In-Memory architecture for a Simple and Linear Fast Adder is described that is poised to revolutionize applications in AI and other domains relying on intensive matrix multiplication. This scalable, cost-effective design, coupled with enhanced speed and energy efficiency, positions this circuit as a foundation for many advancements in the field of computational hardware.

The organization of this proposal is as follows. First, the historical context of set-theoretic axiomatizations of arithmetic and analysis is discussed. Then, addition of natural numbers is described in terms of the proposed set theory. The construction of natural numbers here proposed can be generalized to provide a simple and transparent construction of real numbers that is also compatible with the SLFA. A more detailed presentation of the set theoretical base, its extension to real numbers, other theoretical results, and the SLFA patent description with simulations, are found in [1], [2]. A series of applications to Computer Science follow from this set theoretical proposal. Here, we will only discuss a Simple and Linear Fast Adder for a Compute-In-Memory architecture, and how it can be used for scalar and matrix multiplication. Other applications of this set theory include an analog computing scheme based on the superposition of coupled waves, a fast derivative approximation compatible with the SLFA, Homomorphic Encryption, among others. The theoretical foundation also extends to algebra where finite functions and permutations are given a natural order and a Canonical Block Form for Finite Groups is defined that determines the groups' automorphisms. In analysis, real numbers are constructed in a simple and transparent manner and a fast approximation algorithm is given for calculating the numerical derivative. A Theory of Types that categorizes all mathematical objects with the minimum possible type has applications to Data Structures.

II. Hilbert's Axiomatic Questions and the 24th Problem

There is a widely held belief that the specific selection of a framework for natural [3], and real numbers [4], [5], [6], [7], is inconsequential for the broader field of mathematics. A canonical set theory has been proposed [1], [2] that produces transparent proofs and new results tying fundamental areas of mathematics including group theory, discrete mathematics, analysis, data types, and addressing Hilbert's 24th (twenty-fourth) Problem and Benacerraf's Identification Problem. Computer science applications include a linearly scalable circuit designed for parallel addition and multiplication of scalars, vectors, and matrices, with significant implications for Area-Specific Integrated Circuits (ASICs) and other areas reliant on rapid and low-power vector operations. This In-Memory architecture, based on a patent-pending Simple and Linear Fast Adder (SLFA - patent pending), is a direct consequence of the proposed set theory.

In contemporary mathematics and computer science, it is widely accepted that the natural number zero (0) is represented by the empty set $\emptyset$. However, since the time of Hilbert and even in the present, many mathematicians do not find it necessary to justify the existence of the number 0. It is widely believed that the nature of numbers is irrelevant; what matters is their behavior. In his famous list of 23 Problems, Hilbert paid special attention to the philosophical and practical implications of the Axiomatic Method. Problems 1, 2, and 6 were axiomatic in nature, focusing on the Continuum Hypothesis, the consistency and completeness of arithmetic, and the axiomatization of physics, respectively. Abundant material on the subject of Hilbert and his Axiomatic Program is found in [8] and other articles and books from the same author. In the following decades, a collective effort was made to address these metamathematical questions through the formalization of various set theories. Gödel, Turing, and Church demonstrated the limitations of arithmetical completeness, consistency, and decidability. Gödel's incompleteness theorems, based on Peano's Axioms, shattered illusions of achieving completeness and consistency. And, it was also shown that the Continuum Hypothesis and its negation were both consistent with Zermelo-Fraenkel Set Theory.

Despite the consensus that the choice of axiomatic base for natural numbers and the specific computable coding of other structures is irrelevant, foundational definitions have remained under scrutiny. Philosophers of mathematics, proponents of structuralism, and set theorists have raised questions about the possibility of alternative axiomatic systems to Peano Arithmetic that could address foundational questions in mathematics. Benacerraf's Identification Problem [9], for example, challenges the notion that numbers are sets. He argues that sets are defined to exhibit desired properties. Sets are defined to have the properties we wish them to have. The nature of numbers is projected *onto* sets, but that does not imply a number *is* a set. For instance, Zermelo-Fraenkel Set Theory defines $2 = \{\{\emptyset\}\}$ but Von-Neumann Set Theory defines $2 = \{\emptyset, \{\emptyset\}\}$. Both ways are consistent, and there are in fact infinitely many ways of defining natural numbers. Benacerraf concluded that the number 2 was not any set in particular. Hilbert also analyzed this problem, with a different conceptual approach. But he was also aware of the confusions this kind of question would cause in mathematics and thought it was not prudent to include it in his famous list. He knew that simply asking these questions was enough of a complication to mathematics and that his 24th problem [10] could wait. The problem aimed to develop a theory of the method of proof in mathematics, exploring the idea that under given conditions, there can be only one simplest proof. *"The 24th problem in my Paris lecture was to be: Criteria of simplicity, or proof of the greatest simplicity of certain proofs. Develop a theory of the method of proof in mathematics in general. Under a given set of conditions there can be but one simplest proof."*

III. Theoretical Background

A set theory has been proposed [1], [2] that simplifies proofs in various fundamental areas of mathematics, and provides applications in pure and applied mathematics. This set theory is proposed as a unique canonical set theory that forms an optimal universe for classical mathematics from number theory to analysis [1]. The order of natural numbers, and the operations of addition and multiplication, are defined on the set of all Hereditarily Finite Sets, **HFS**, in a novel manner that treats numbers as sets and not sequences, with important consequences in the representation and classification of finite and infinite objects that will not be discussed here [2]. Algebraic invariants are described with results bringing together set theory, discrete mathematics, number theory and algebraic structures. The proposed construction of natural numbers is generalized to provide a simple and transparent construction of the continuum of real numbers, with a fast approximation for the numeric derivative that can be implemented with the SLFA. Infinite data structures are defined in the most efficient way with the smallest possible data type, using meaningful and computable codings. In general, all mathematical objects are well assigned to tree structures.

This paper will focus on the Ackermann Coding which is a bijection $\mathbb{N} \to \textbf{HFS}$, also known as BIT-Predicate [11], that maps the natural number $\sum_i 2^{x_i}$ to the set $\{x_1, x_2, \ldots, x_n\}$. Addition is a special prefix problem which means that each sum bit is dependent on all equal or lower input bits [12], [13]. Although solutions to the carry-over delay of the addition algorithm exist in practice, the problems related to the circuitry's complicated topology and gate depth quickly outweigh the benefits as the number of bits is slightly increased. The complexity associated to a non-scalable design increases the cost of design and production. Energy efficiency and delay are also associated to gate depth.

A. Adding Sets

An addition algorithm for BIT-Predicate is described, as a finite state machine of logarithmic time, that can be implemented as a Simple and Linear Fast Adder. To add two sets

A, B, form two new sets $A' = A \triangle B$ and $B' = s(A \cap B)$, where $A \triangle B$ is the symmetric difference. The function s adds 1 to the elements of its argument. The addition of these two new sets is the same as the original addition $A \oplus B = A' \oplus B'$ because the powers of 2 in $A \oplus B$ have only been rearranged. The term A' consists of the non repeated powers (symmetric difference). The term B' is a displacement of the repeated powers; it is a function of the intersection. In a finite number of iterations the intersection $A^{(k)} \cap B^{(k)} = \emptyset$ is reduced to the empty set. The final result is $A^{(k+1)}$, because $A \oplus B = A^{(k+1)} \oplus B^{(k+1)} = A^{(k+1)} \oplus s(\emptyset) = A^{(k+1)}$.

Let us apply this reasoning with an example, calculating $13 + 25 = 38$. The addition is rewritten as the sum of sets $A \oplus B = \{0, 2, 3\} \oplus \{0, 3, 4\}$ because $13 = 2^0 + 2^2 + 2^3$ and $25 = 2^0 + 2^3 + 2^4$. The inputs will be represented with two side-by-side vectors.

$$\begin{array}{cc} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 0 \\ 1 & 1 \end{array}$$

Then, two new columns are formed. The column on the left will take the value 1 everywhere in the symmetric difference. The right column will take the value 1 everywhere in the intersection, but the values need to be displaced one unit up corresponding to the fact the intersection is representing addition of two equal powers of 2 and $2^n + 2^n = 2^{n+1}$. This gives the new columns

$$\begin{array}{cc} 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0. \end{array}$$

Iterating this step gives the next columns

$$\begin{array}{cc} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 0. \end{array}$$

In the next iteration the system stabilizes because the intersection is now the empty set, and if the process is iterated again it will give the same configuration.

$$\begin{array}{cc} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 0. \end{array}$$

The process described herein is a finite state machine. A finite configuration of particles in a column represents a natural number, and each state is a pair of natural numbers. In general, the left column of the next state is given by the energy levels not repeated in the current state. The right column of the next state is given by a displacement, one level up, of the energy levels repeated in the current state. A stable state is reached in a finite number of steps. This happens when the right column is empty; the result of the sum is given in the left column. The average number of steps for adding two n-bit numbers is $\max(n)$, while the total number of steps is bounded by n. The probability of reaching stability in $i \leq n$ steps is the probability of obtaining i-many consecutive heads in a trial of n-many fair coin tosses; the probability of reaching stable state in i-many steps decreases dramatically as i moves further from $\max(n)$.

B. Addition of Multiple Inputs

A general method for defining the sum of multiple operands is proposed that has several advantages in hardware implementation. An algorithm is described that reduces the sum of k summands to $\max(k) + 1$ summands. Consider the sum of 4-many, 8-bit numbers. Let $A = a_0 a_1 \cdots a_7$, $B = b_0 b_1 \cdots b_7$, $C = c_0 c_1 \cdots c_7$, $D = d_0 d_1 \cdots d_7$.

$$\begin{array}{cccc} a_7 & b_7 & c_7 & d_7 \\ a_6 & b_6 & c_6 & d_6 \\ a_5 & b_5 & c_5 & d_5 \\ a_4 & b_4 & c_4 & d_4 \\ a_3 & b_3 & c_3 & d_3 \\ a_2 & b_2 & c_2 & d_2 \\ a_1 & b_1 & c_1 & d_1 \\ a_0 & b_0 & c_0 & d_0, \end{array}$$

where each a_i, b_i, c_i, d_i takes a value in $\{0, 1\}$.

There is a total of four columns so that three bits are sufficient for counting how many 1's are contained in a single row because $\max(4) + 1 = 2 + 1 = 3$. The number of 1's in each row can be represented in a 3-column grid with $8 + (3 - 1) = 10$ many rows.

$$\begin{array}{ccc} 0 & 0 & c'_9 \\ 0 & b'_8 & c'_8 \\ a'_7 & b'_7 & c'_7 \\ a'_6 & b'_6 & c'_6 \\ a'_5 & b'_5 & c'_5 \\ a'_4 & b'_4 & c'_4 \\ a'_3 & b'_3 & c'_3 \\ a'_2 & b'_2 & c'_2 \\ a'_1 & b'_1 & 0 \\ a'_0 & 0 & 0 \end{array} \qquad (1)$$

The elements a'_0, b'_1, c'_2 will be used to write the number of 1's in row 0. The elements a'_1, b'_2, c'_3 are used to write the number of 1's in row 1, and elements a'_2, b'_3, c'_4 are used to write the number of 1's in row 2, etc. This maintains the representation of energy-levels and their unit value, while avoiding any intervention with totals from one row and another. The three column grid can be reduced to two columns, by iterating the process. The total number of units in each row of (1) will be represented with two bits because $\max(3)+1 = 1+1 = 2$. Addition of the two columns

$$\begin{array}{cc}
a''_9 & b''_9 \\
a''_8 & b''_8 \\
a''_7 & b''_7 \\
a''_6 & b''_6 \\
a''_5 & b''_5 \\
a''_4 & b''_4 \\
a''_3 & b''_3 \\
a''_2 & b''_2 \\
a''_1 & b''_1 \\
a''_0 & 0
\end{array}$$

is equivalent to the original four-input addition. Elements a''_0 and b''_1 represent the total value of the first row in (1). Elements a''_1 and b''_2 represent the total value of the second row, elements a''_2 and b''_3 represent the total value of the third row, etc. An additional row is not required because row 10, in (1), at most will contain a single 1, requiring one row.

Let $A = 32 = \{6\}, B = 36 = \{2,5\}, C = 22 = \{1,2,4\}, D = 36 = \{2,5\}, E = 26 = \{1,3,4\}, F = 15 = \{0,1,2,3\}, G = 15 = \{0,1,2,3\}, H = 51 = \{0,1,4,5\}$. This is given by

$$\begin{array}{cccccccc}
1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1.
\end{array}$$

There can be at most eight objects in each row, so only four bits are needed to represent the total of number of 1's in a single row. This means the new grid has four columns. There is a total of 3-many 1's in row 0. This is represented by placing the sequence of digits 1100 (larger powers are to the right and smaller powers are to the left so we have 1100 representing 3, instead of 0011) in the bottom most diagonal, of the new grid.

$$\begin{array}{cccc}
 & & & 0 \\
 & & 0 & 0 \\
 & 1 & 0 & 0 \\
1 & 0 & 0 & 0.
\end{array}$$

The next row, row 1 has 5-many 1's. The sequence of digits 1010 goes in the next diagonal up.

$$\begin{array}{cccc}
 & & & 0 \\
 & & 1 & 0 \\
 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0.
\end{array}$$

There are also 5-many 1's in row 2, so the sequence 1010 is placed in the next diagonal up.

$$\begin{array}{cccc}
 & & & 0 \\
 & & 1 & 0 \\
 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0.
\end{array}$$

Rows 3 and 4 have 3-many 1's each, so the sequence 1100 is placed in each of the following diagonals up.

$$\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0.
\end{array}$$

Row 5 has 4-many 1's, so the sequence 0010 in the last diagonal.

$$\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0.
\end{array}$$

These four columns can be reduced to three columns, by iterating this process. The bottom-most row of the last grid, has a total of 1 number 1's so that the sequence 100 is placed on the bottom diagonal.

$$\begin{array}{ccc}
 & & 0 \\
 & 0 & 0 \\
1 & 0 & 0.
\end{array}$$

There is a total of 2 number 1's in the next row, row 1, so that the sequence 010 is placed on the next diagonal.

$$\begin{array}{ccc}
 & & 0 \\
 & 1 & 0 \\
0 & 0 & 0 \\
1 & 0 & 0.
\end{array}$$

Continuing in this manner gives

$$\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
1 & 0 & 0 \\
0 & 0 & 0 \\
1 & 1 & 0 \\
1 & 1 & 0 \\
0 & 0 & 0 \\
1 & 1 & 0 \\
0 & 0 & 0 \\
1 & 0 & 0.
\end{array}$$

Using the same method, these three columns are reduced to two columns, by the same formula $\max(3) + 1 = 2$.

$$\begin{array}{cc}
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0 \\
0 & 0 \\
0 & 1 \\
0 & 1 \\
0 & 0 \\
0 & 1 \\
0 & 0 \\
0 & 0 \\
1 & 0.
\end{array}$$

Unsurprisingly, the method employed for reducing columns coincides with the Finite State Machine that adds two set numbers. The column reduction method will reduce the addition of two columns to the addition of $\max(2) + 1 = 1 + 1 = 2$ columns, but it will reach a stable state in a finite number of iterations. In this sense, the Finite State Machine for adding two numbers is a particular case of a more general FSM that reduces the addition of k-many columns to the addition of $\max(k)+1$ columns. The Final result of the four input addition is equal to 233.

$$\begin{array}{cc}
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0 \\
0 & 0 \\
1 & 0 \\
1 & 0 \\
0 & 0 \\
1 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0.
\end{array}$$

IV. Simple and Linear Fast Adder of Two Inputs (Patent Pending)

Traditional adders face the challenge of eliminating propagation delays in an efficient manner. While existing methods attempt parallel and fast addition, they often incur drawbacks such as increased area, elevated energy consumption, and heightened costs in design and manufacturing. Established approaches like Carry Look-Ahead (CLA) adders and other fast adders [14], [15], [16], [17] have their circuit complexity, area and gate count/depth significantly enlarged with each added bit.

The SLFA has very low gate count and constant depth. It is compatible with natural numbers, integers, and rational numbers. Additionally, the SLFA is linear in topology and linearly scalable. Its architecture is analogous to that of the box cars of a train. Adding a bit of input is simply adding a box car. This is true also with the traditional Carry-Over adders, which also consist of half adders connected in series. Thus, the SLFA enjoys of the benefits of both types of adders. It has the simple and linear topology of carry-over adders, but its performance is also projected to be at least as good as other fast/parallel adders whose complexity grows fast. Adding a bit of input to the circuit amounts to connecting a subunit to the end of the circuit. Each sub unit consists of a half adder and a 2-bit register.

The Simple and Linear Fast Adder (SLFA) is structured with n subunits connected in series, where each subunit comprises a half adder and two memory registers as depicted in Fig. 1. The registers are double-edge triggered, reading on the rising edge and writing on the falling edge. Initially, inputs A and B are stored in registers RA and RB, respectively. Subsequently, the contents of these registers follow two paths. The inputs of each specific significant bit undergo 'XOR' and 'AND' operations. The 'XOR' gate outputs the symmetric difference, which is then directed to register RA in the same significant bit. The 'AND' gate outputs the intersection, sent to register RB in the next significant bit. A bit shift is applied to the intersection, representing repeated powers of 2. This iterative process continues until register RB becomes the zero vector. The SLFA executes the sum of two numbers in logarithmic time, on average. Increasing the number of bits is equivalent to adding one subunit consisting of two memory elements and one half adder. Gate depth is that of a zero flag of n-bits whose input are the contents of RB and signals when the input is the zero vector. Time delay will be shorter and the SLFA will be more energy efficient because gate depth and instruction set are constant. The cost of design and production [18] will also be lower because of its simple and linear topology.

Let us understand this adder with examples. Suppose we wish to add $3 + 1 = 4$. We would first load the inputs by setting $RA = 0011$ and $RB = 0001$. Then, running the SLFA one iteration will give the new configuration $RA = 0010$ and $RB = 0010$. Running the SLFA again gives $RA = 0000$ and $RB = 0100$. The next iteration of the SLFA gives $RA = 0100$ and $RB = 0000$. Then, on the next step, the process is terminated because RB signals the zero vector. The complete patent description, detailed examples, figures, claims, are found in [2].

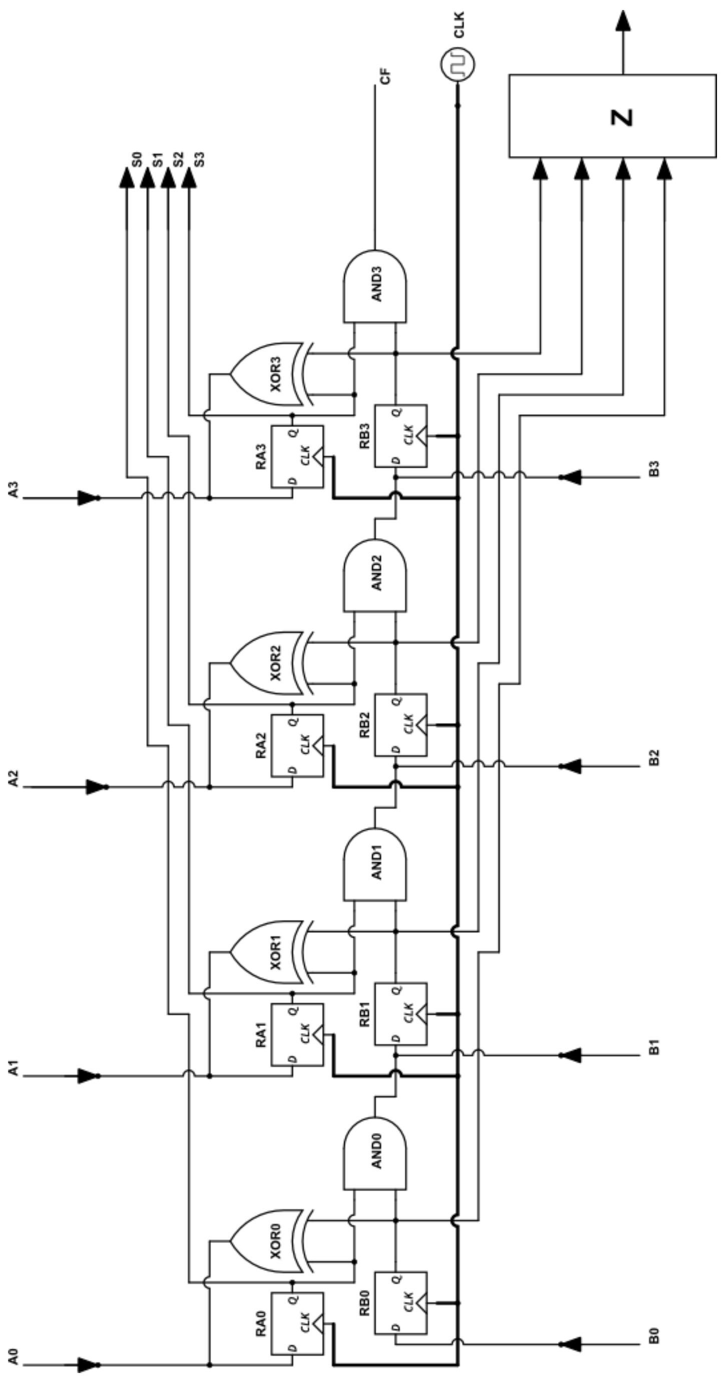


Fig. 1. A 4-bit SLFA, consisting of 4 subunits connected in series. Each subunit is made up of a half adder and a 2-bit register. The ‘XOR’ gates find the symmetric difference, while the intersection is found in the output of the ‘AND’ gates. A bit shift is applied to the intersection, and the process is iterated until register RB is the zero vector.

V. Compute-In-Memory Implementation for Addition of Multiple Inputs

The SLFA circuit, as we will see in this section, can also be arranged in a rectangular matrix, that is capable of adding multiple inputs and multiplying two numbers. The size of the rectangular matrix is also trivially scaled to operate more bits or inputs. This scalability provides material benefits for matrix multiplication, which will be discussed in this section. Most importantly, it represents a transformative solution to the Von Neumann bottleneck. The SLFA is a linear architecture of subunits connected in series, with each subunit consisting of 2-bits of memory and a half adder. The rectangular array for addition of multiple inputs, which will be detailed below, is a rectangular version of the SLFA. Each node of this rectangular grid consists of a half adder and a 3-bit memory register. Some nodes of the same column or row are connected. Processing is done in one rectangular layer, and memory registers are in a second, topologically equivalent, layer. Every node in one layer is connected with the corresponding node in the other layer. That is to say, layers are connected in the trivial one-to-one manner. This cuts data migration requirements because data only has to be moved the distance between layers, instead of peripheral as in the Von-Neumann Architecture. By mitigating drawbacks associated with parallel adders and Von Neumann architecture, the SLFA presents a shift towards more efficient and versatile computing architectures of unparalleled efficiency and performance, that can more easily be scaled past 64-bit architecture. The Written Opinion on patentability from the International Search Authority (ISA/US) is provided in the appendix, at the end in Fig. 6.

Consider b-many n-bit SLFAs side by side, as in Fig. 2. A small set of parallel connections (only nodes from the same row or column are connected) allow for the rectangular circuit to perform addition of n-many b-bit numbers. Fig. 2 is illustrative of the topology of the multiple input adder, but it is not an accurate depiction of the multiple-input adder. Each node of Fig. 2 will consist of at least one 3-bit register and at least one half adder, maintaining the gate count and depth very low. Every row is a SLFA, and only parallel connections between nodes of the same row or column are needed. In a given node, two of the registers are part of the SLFA of that row. The third memory bit will be referred to as the principal bit. The principal bits of the nodes store the initial inputs. The principal bits in a given column represent an input of b-bits. There is a total of n columns so that the adder will be able to perform addition of n inputs. There are two types of connections on nodes of the same row. The first kind (thinnest lines of Fig. 2) correspond to connections of the SLFA conforming that row. The second kind (thick lines) is needed for counting the number of 1’s stored in the principal bits, by sending the contents of the principal bits to the least significant bit of the SLFA of that row. Connections between elements of the same column (thickest lines) correspond to the diagonal placement of the total number of 1’s in a given row, of the column reduction algorithm.

Each SLFA will count the number of 1's in its row. To achieve this, the contents of the principal bits in a row must be sent to the least significant bit of register RB in the SLFA of that row. The objects of a row have to be counted one-by-one. That is to say, first you send the left-most principal bit of the row, to register RB_0 of the SLFA. Then, you count the next object, then the next, and so on until you have counted them all. This is done in parallel, for all rows, so that all rows are counted simultaneously. Each SLFA can operate independently and signal process termination individually. This is important because each row will take a different number of iterations and each iteration will consist of a different number of subprocesses, each subprocess taking a different number of steps, to count how many 1's are in that row. Once the elements of a row are counted, the results are stored in register RA of the SLFA. When all rows have been counted the contents of registers RA will be sent to their new principal bits. This is achieved with the connections between nodes of the same column. The first column is connected with itself. In the next column, every node will be connected to the next node up. In the third column, each node will be connected with the second node up, and so on. These connections on nodes of the same columns will enable the diagonal placement of the row count. In each iteration the columns with non-zero entries in their principal bits are less. This process can be iterated until only the first column is a non-zero vector in the principal bits.

The first step is to count the number of 1's in each row. Given that the SLFAs conforming the rows operate independently, all rows will be counting simultaneously. Start by sending the principal bits of the first column, each to its corresponding register RB_0. Then, run the SLFAs (using thin lines connecting nodes of the same row) so that at the end of the cycle, registers RA hold the count. Next, send the contents of the principal bits from the second column to the least significant bit of their respective SLFA, saving it in register RB_0. This is done using the second kind (thick lines) of connections on nodes of the same row. Run the SLFAs again. Register RA of each row holds the count of 1's in the first two inputs of that row. At this point there are three possibilities for the state of RA, given a fixed row. There can be either $0, 1$ or 2-many 1's counted so far, in each row. If there are zero objects counted so far in a row, then register RA of that row has $RA_0 = RA_1 = 0$. If one object has been counted in a given row, then that row has $RA_0 = 1$ and $RA_1 = 0$. Finally, if two objects have been counted in a row, then register RA of that row's SLFA will have $RA_0 = 0$ and $RA_1 = 1$. Next, send the contents of the principal bits in the third column to RB_0 of their respective SLFA. The SLFAs are run again to find the total number of 1's in the first three elements of each row. Continue in this manner for all columns. After running the SLFAs for the last column, each SLFA contains the total count of 1's in that given row. The total count of 1's in a given row is stored in register RA of that SLFA. Recall that each row is independent, so that while one row may be counting the fifth bit of that row, another row may be working on adding the seventh bit. Once all rows are done counting the number of 1's and each row has its total count stored in its register RA, the next step is to send the contents of registers RA to the principal bits corresponding to the diagonal placement of the total count of 1's. This happens for the b-many registers RA, simultaneously. This is achieved through the connections between nodes of the same column (thickest lines). This step is equivalent to the diagonal placement of the count of 1's in a row, in the column reduction algorithm. Now, the entire process is iterated, but the number of input columns has been reduced to $\log n$ columns.

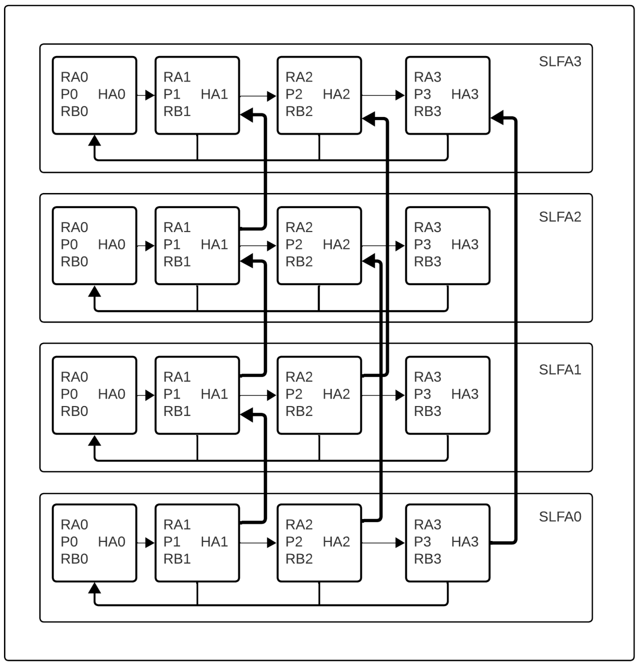


Fig. 2. A $b \times n$ grid of subunits constitutes an adder of n-many b-bit numbers. This rectangular array is a simplified representation of a circuit that can simulate the column reduction algorithm for adding multiple inputs. Every column is a b-bit input. Each row consists of an n-bit SLFA, and every node of the SLFA has an extra bit, called the principal bit. Every SLFA conforming a row, will count the number of 1's in that row. Then, the results are sent to principal bits of the same column, in a diagonal manner that simulates the column reduction algorithm. Here, a 4×4 grid is shown.

Let us analyze this circuit through examples. Fig. 2 is a 4×4 grid so this has to be taken into account for overflow instances. First, let us carry out the operation $2 + 3 + 2 + 3$. For this, we have to load the inputs to the principal bits. The first input $2 = 0010$ will be uploaded to the left-most column, column 0. This means that P_0 in SLFA1 of Fig.2 has the value "1" stored. The other principal bits P_0 of column zero 0 will store the value "0". The next column, column 1, is going to store the value "1" in the principal bits P_1 of SLFA0 and SLFA1, while "0" is stored in the principal bits P_1 of SLFA2 and SLFA3. The configuration of column 2 is the same as column 0, and the configuration of column 3 is identical to column 1. The inputs of the principal bits will appear as in the matrix below.

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1. \end{matrix}$$

Once the principal bits have stored the inputs, the number of 1's in each row must be counted. The bits in column 0, the left-most column, will be counted first. Each of the four SLFA has a principal bit P_0. Those bits are sent from P_0 to register RB_0 of the same SLFA; they are displaced within their same node. At the end of this step, SLFA1 will have $RB_0 = 1$. All other values RA_i and RB_i in the SLFAs are set to 0. We run the SLFAs. This time, all four of the SLFAs will start the addition process, but as more iterations are made, each SLFA will be running independently. Once a row has finished a subprocess (the addition corresponding to adding a 1/0 from that row), it will request the next bit of that row, independently of the other rows. Let us first simulate what happens in SLFA0,SLFA2 and SLFA3. Register RB in those rows will signal termination of the addition subprocess because they are equal to the zero vector. This means that in the next clock cycle, SLFA0,SLFA2 and SLFA3 will be receiving the next bit of their row, to add. Row 1, on the other hand will send the contents of its registers RA and RB through the half adder and RB will not signal subprocess termination because it is not the zero vector. The result at the end of this step will be that the value of 1 that was stored in RB_0, is now stored in RA_0. In the next iteration, register RB_0 of SLFA1 will be the zero vector so that subprocess termination will be signaled. SLFA1 will receive the next principal bit of its row, P_1, one step after the other SLFAs. SLFA1 signals subprocess termination of adding the first significant bit of its row, when the other adders receive the next significant bit to count. That is to say, SLFA1 signals the termination of adding the first bit, in the same step that the other adders are migrating the contents of P_1 to RB_0. At the end of this step, SLFA1 has $RA_0 = 1$ and $RB_0 = 0$. On the other hand, SLFA0 now has $RA_0 = 0$ and $RB_0 = 1$ because it has imported the contents of $P_1 = 1$ to RB_0 for counting. All other values of the SLFAs are zero. In the next cycle, SLFA1 imports the next bit, $P_1 = 1$, into register RB_0 for counting. This means that the configuration of SLFA1 will be $RA_0 = RB_0 = 1$ and $RA_1 = RB_1 = 0$. Meanwhile, SLFA0 sends the bit stored in RB_0 to RA_0. The configuration of SLFA0 at the end of this step is $RA_0 = 1$ and $RB_0 = RA_1 = RB_1 = 0$ meaning that it has counted a single 1 in the first two bits of that row. Simultaneously, SLFA1 will send the contents of RA and RB through the half adders and at the end of this step this adder will have $RA_0 = RB_0 = RA_1 = 0$ and $RB_1 = 1$ which means that SLFA1 has counted two 1's in the first two bits of its row. SLFA0 will signal subprocess termination and request the next bit, P_2, because its register RB is the zero vector. During this step, SLFA1 will send the contents of RA and RB through the half adder, resulting in $RA_0 = RB_0 = RB_1 = 0$ and $RA_1 = 1$. Next, SLFA0 will store the contents of $P_2 = 0$ in RB_0, at the same time that SLFA1 signals termination of the addition subprocess. In the next iteration, SLFA0 signals termination and requests the contents of P_3, while SLFA1 saves the contents of $P_2 = 1$ in its register RB_0. In the next step, SLFA0 will move the contents of $P_3 = 0$ into RB_0. The configuration of SLFA0 is now $RA_0 = RB_0 = 1$ and $RA_1 = RB_1 = 0$. In this same step, running SLFA1 moves the 1 saved in RB_0 to RA_0. This leaves the configuration of SLFA1 as $RA_0 = RA_1 = 1$ and $RB_0 = RB_1 = 0$, meaning that it has counted a total of three 1's in the first three bits of row 1. The subsequent step will make SLFA0 send its contents through the half adders and the new configuration of its registers will be $RA_0 = RB_0 = RA_1 = 0$ and $RB_1 = 1$. This means that SLFA0 has counted a total of two 1's in the first three inputs of that row. Meanwhile, in SLFA1, the subprocess of addition is signaled as terminated because RB is read as the zero vector. Next, SLFA0 will send its contents through the half adders, and the new configuration is $RA_0 = RB_0 = RB_1 = 0$ and $RA_1 = 1$. In this same step SLFA1 moves the contents of P_3 to RB_0. After this, in the next iteration, SLFA0 signals termination because its register RB is the zero vector. This means that SLFA0 is already done counting because it has added all the bits of that row, from P_0 thru P_3, and it will standby until SLFA1 is done counting. The total count of 1's in the principal bits of row 0 is given in RA of SLFA0 as $RA = 0010$ because $RA_0 = RA_2 = RA_3 = 0$ and $RA_1 = 1$. The current state of SLFA1 is given by $RA_0 = RB_0 = RA_1 = 1$ and $RB_1 = 0$. Running the SLFA will reconfigure the registers to $RA_0 = RB_0 = 0$ and $RA_1 = RB_1 = 1$. Another iteration of SLFA1 will give $RA_0 = RB_0 = RA_1 = RB_1 = 0$ and $RB_2 = 1$. One more iteration of SLFA1 gives $RA_0 = RB_0 = RA_1 = RB_1 = RB_2 = 0$ and $RA_2 = 1$. In the next step, RB of SLFA1 is the zero vector so that in the step after that, the addition of P_3 is signaled as finished. The total count of 1's in row 1 is given in register RA. Thus, we can conclude that there are four 1's in row 1 because $RA = 0100$. Now that we have the total count of 1's in each row, these totals have to be sent to the corresponding bits, as given by the column reduction algorithm. The connections between nodes of the same columns are used for sending the contents of registers RA to predetermined principle bits. Let us first observe that SLFA0 has a 1 stored in RA_1. The column connections (thickest lines) will send this value 1 to the principal bit P_1 of SLFA2. At the same time, the 1 saved in RA_2 of SLFA1 is going to be sent to the principal bit P_2 of SLFA3. This means that we have reduced the addition of four numbers, to the addition of three numbers. The new columns are given by the matrix

$$\begin{matrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0. \end{matrix}$$

Counting the number of 1's in each row of the last matrix will give us two new columns.

$$\begin{matrix} 1 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0. \end{matrix}$$

This process has to be iterated until only the first column is different from the zero vector, so that we are done. The result is equal to ten $2+3+2+3 = 1010$, as we would expect.

When the i-th significant bit of a row is sent to the SLFA, the addition that follows takes at most $\max(i)+1$ iterations of the SLFA. The worst case scenario, when adding n inputs, occurs if a row has n-many $1's$ in the principal bits. The number of steps in the worst case scenario is bounded by $\max(2)+\max(3)+\max(4)+\ldots+\max(n)+n$, but is much lower because most of the terms can be bounded by smaller numbers. For example, if i is a multiple of 2, then only one iteration of the SLFA is needed in the i-th step. If i is odd, then at least one iteration is needed. The only occasions where $\max(i)+1$ iterations of the SLFA are needed is if i is of the form 2^k-1, for some k.

The relative efficiency of this implementation with respect to other circuits could be analyzed by cases, in terms of the quotient of n and b. For large b and small n, it is easy to see the advantages this circuit would have because it calculates the total number of elements in a row, and it does all rows in parallel. The more rows there are relative to columns the more advantage provided by this method. There is a problem when n gets too big. When the i-th bit of a given row is sent to the SLFA for counting, the SLFA performs $\max(i)+1$ many iterations. If n is too large this method will present diminishing returns, as i approaches n. However, for this case there is an alternative. The number of summands can be reduced by half in a fixed number of steps. The method reduces the addition of n summands to the sum of $\max(k)+1$. Thus, 8 summands can be reduced to $\max(8)+1=4$ summands. If the number of summands is a multiple of 8, then this fact can be used to reduce by half the number of summands. There is another way to reduce summands by half because 4 summands are reduced to $\max(4)+1=3$ summands which in turn are reduced to $\max(2)+1=2$ summands. This means that if the number of summands is a multiple of 4, then the number of summands can be reduced to half in this manner. These alternate methods of reduction into half are achieved by rearranging the vertical and horizontal connections of the grid. Depending on the quotient and size of n and b there will be an optimal size for reduction of summands that minimizes time complexity, and the topology of the nodes is unchanged.

The Von Neumann Architecture conventionally segregates memory and ALU into distinct components, necessitating continuous data migration between these units. This architectural division, known as the Von Neumann bottleneck, is the biggest contributor to energy consumption and time delays in a processor. The inherent challenge arises from the rigid, rectangular grid structure of memory, conflicting with the intricate and irregular connections inherent in logic circuitry for arithmetic operations. To address these conflicts, a novel approach is taken by allocating memory units to one layer and logic circuitry to another layer [19] as in Fig. 3. One layer is a matrix of memory nodes with size $b\times n$; each node containing at least a 3-bit register. A second layer is a logic grid of the same size $b\times n$ each node comprised by at least one half adder. This strategic separation is possible due to the identical rectangular geometries of both layers and an equal number of nodes in one-to-one correspondence. This superposition of two rectangular grids of equal size solves fundamental challenges and lays the groundwork for Computing-In-Memory [20]. The resulting architecture promises noteworthy improvements in terms of reduced delay and energy consumption. This innovative approach warrants further exploration and comparison [18] against alternative architectures to comprehensively assess its advantages and potential applications. Other multipliers can be found in [21], [22], [23].

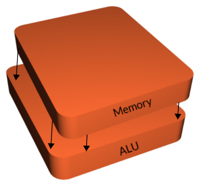


Fig. 3. The multiple input adder can be organized in two layers. One layer is reserved for memory registers, and the other layer is a grid of half adders.

A. Scalar Multiplication

Given that this low-powered circuit performs addition of multiple inputs, it is compatible with parallel-multiplication of two inputs. Additionally, the circuit is able to perform parallel addition of b-many pairs of n-bit numbers, because each SLFA (rows) can be used as an independently-timed SLFA. The circuit is linearly scaleable in terms of bits and inputs, it presents the minimum possible topological complexity (rectangular grid of nodes with parallel connections), and is low-powered due to the gate depth. Although PASTA adders [24] are topologically equivalent to the SLFA, it is important to note the PASTA adder is an asynchronous circuit, like most fast-adders, and therefore it lacks the memory units necessary for this addition of multiple inputs. The PASTA adder has multiplexers instead of the registers used in the SLFA, making it inappropriate for In-Memory arithmetic.

B. Matrix Multiplication

There are several benefits in using this architecture for matrix multiplication. Examples of current solutions to matrix multiplication are proposed and referenced in [25].

Let $\mathbf{A},\mathbf{B}$ two matrices of size $p\times n$ and $n\times q$, respectively, and let the elements of the matrix be b-bit numbers. Let M_1 a rectangular array of size $2bn\times b$. Rows 1 through $2b$ form an adder of b-inputs of b-bits. Rows $2b+1$ through $4b$ form another adder of b-inputs of b-bits, etc. Array M_1, of Fig. 4, functions as n-many, b-input adders of b-bit numbers. It is able to parallelly execute the partial products of the dot product $\mathbf{a} \cdot \mathbf{b}$, where $\mathbf{a} = (a_1, \ldots, a_n)$ is a row vector of $\mathbf{A}$ and $\mathbf{b} = (b_1, \ldots, b_n)$ is a column vector of $\mathbf{B}$. That is to say, M_1 can execute the partial products $a_1 \cdot b_1, \ldots, a_n \cdot b_n$, in parallel. The partial product $a_1 \cdot b_1$ is calculated in rows 1 thru $2b$, product $a_2 \cdot b_2$ is calculated in the next $2b$-many rows, rows $2b+1$ thru $4b$, etc.

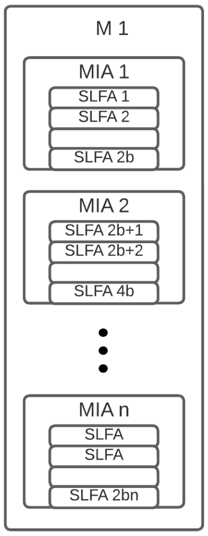


Fig. 4. M_1 consists of n-many multiple input adders, each one able to compute partial product of the dot product.

Suppose you have additional copies $M_2, \ldots, M_q$, of M_1. Let $\mathbf{b}_i$ the i-th column of $\mathbf{B}$ and let $\mathbf{a}$ any row vector of $\mathbf{A}$. Use M_1 for calculating the partial products of $\mathbf{a} \cdot \mathbf{b}_1$, use M_2 for finding the partial products of $\mathbf{a} \cdot \mathbf{b}_2$, etc. Employing $M_1, \ldots, M_q$ allows to compute, in parallel, all of the partial products involved in finding an entire row of $\mathbf{A} \cdot \mathbf{B}$.

If we consider an extra row of adders, call it M_0, it will serve to parallel add all the partial products, Fig. 5. The adders of M_0 are required to be of size $(2b + \max(n)) \times n$. The multiple input adder MIA1 of M_0 will add the partial products of $\mathbf{a} \cdot \mathbf{b}_1$, that were calculated in M_1. Adder MIA2 of M_0 will add the partial products corresponding to $\mathbf{a} \cdot \mathbf{b}_2$, from M_2, etc. This way, using $M_0, M_1, \ldots, M_q$ allows for the parallel computation of an entire row in the time that it takes to multiply two b-bit numbers (calculate partial products), plus the time it takes to add n-many $2b + \max(n)$-bit numbers (add partial products). A single adder M_i, from M, calculates a single element of the product matrix $\mathbf{A} \times \mathbf{B}$. Therefore, taking p-many copies of circuit M, from Fig. 5, will allow for the multiplication of a Matrix in the same time. In matrix multiplication it can often be the case that the number of bits of the result, is much smaller than the number of rows and columns. Adaptations can be made for that and other cases, based on area-specific use and pipeline needs.

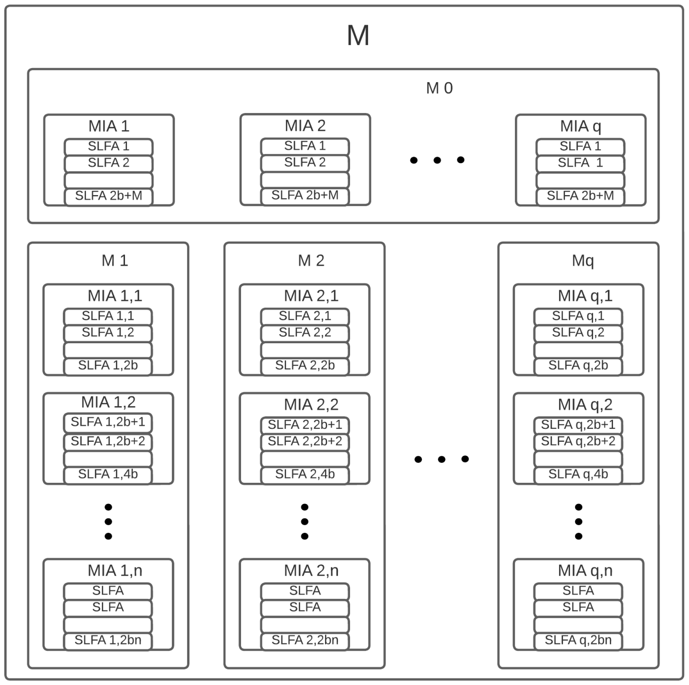


Fig. 5. Using a total of q-many units $M_1, \ldots, M_q$, it is possible to execute the partial products of a row. The multiple input adders of M_0 will add the corresponding partial products. The multiple input adder MIA_i, of M_0, will add the partial products $MIA_{(i,1)}, \ldots, MIA_{(i,n)}$, calculated in M_i.

VI. Conclusion

In a world pushing the boundaries of innovation, we are reaching the limits of current technological frameworks. A novel conceptualization of the foundations of mathematics, with immediate applications in a number of key technologies is proposed, facilitating seamless data representation and computational operations. Information theory and computer science, from high-level Software languages down to the System-on-Chip Hardware level, can be benefited. The Patent-Pending Compute-In-Memory architecture of the Simple and Linear Fast Adder maximizes efficiency and performance of matrix multiplication. Additionally, a Fast Derivative Approximation is compatible with the SLFA. The set theory proposed can unify diverse computer science domains under one robust mathematical framework. This integration blueprint connects different applications for improved standards of next-generation devices and systems, at various levels. It pushes for a Vertical and Horizontal implementation of mathematically optimal solutions, enhancing efficiency and security throughout the universal digital ecosystem. For example, the Fast Derivative Approximation can be performed with the SLFA in two cycles, plus at most n-bit shifts. The scalability and linearity of the circuit topology also allows for easier design of architectures that go beyond 64-bits. State-of-the-art Homomorphic Encryption can be merged together with the arithmetic architecture for designing Encrypted Processing

Units. A range of challenges for HE can be solved in this model, including a version of Homomorphic Encryption that merges the Processing and the Decryption steps, into a single step. The operation is the key, so the second party can only use the encrypted inputs for the intended purposes. Information in the cloud will not have to be stored or accessed in plaintext by a second party, in order for them to use that information. Therefore, in analogy we can think of data shared in this cloud as current, instead of stored files. The state of development of these applications and others can be found at the author's personal page.

References

[1] J. P. Ramírez, "A New Set Theory for Analysis," Axioms. 2019, 8, 31.

[2] J. P. Ramírez, "Canonical Set Theory with Applications from Parallel Matrix Operations and Data Structures to Homomorphic Encryption," (Preprint) Author's homepage: www.binaryprojx.com. 2023.

[3] P. Bernays, "Axiomatic Set Theory," Dover: New York, NY, USA, 1991.

[4] N. A'Campo, "A Natural Construction for the Real Numbers," arXiv, 2003; arXiv:math.GN/0301015 v1.

[5] R. D. Arthan, "The Eudoxus Real Numbers," arXiv, 2004; arXiv:math/0405454.

[6] N. G. De Bruijn, "Definig Reals Without the Use of Rationals," Koninkl. Nederl. Akademie Van Wetenschappen: Amsterdam, The Netherlands, 1976.

[7] A. Knopfmacher and J. Knopfmacher, "Two Concrete New Constructions of the Real Numbers," Rocky Mt. J. Math. 1988, 18, 813–824.

[8] L. Corry, "David Hilbert and the Axiomatization of Physics (1898–1918): From Grundlagen der Geometrie to Grundlagen der Physik," Springer Netherlands, 2010.

[9] P. Benacerraf, "What Numbers Could Not Be," Philos. Rev. 1965, pp.74.

[10] R. Thiele, "Hilbert's Twenty-Fourth Problem," The American Mathematical Monthly, 110:1, 1-24, **2003**. DOI: 10.1080/00029890.2003.11919933

[11] W. Ackermann, "Die Widerspruchsfreiheit der allgemeinen Mengenlehre," Math. Ann. 114, 305–315.

[12] R. E. Ladner and M. J. Fischer, "Parallel Prefix Computation," Journal of the ACM, 27(4), pp. 831-838, October 1980.

[13] N. Metropolis, G. C. Rota, S. Tanny, "Significance Arithmetic: The Carrying Algorithm," Journal of Combinatorial Theory, Series A, 1973, 14, 386–421.

[14] R. Uma, V. Vijayan, M. Mohanapriya, S. Paul, "Area, Delay and Power Comparison of Adder Topologies," International Journal of VLSI design & Communication Systems (VLSICS) Vol.3, No.1, February 2012.

[15] R.P.P. Singh, P. Kumar, B. Singh, "Performance Analysis Of Fast Adders Using VHDL," 2009 International Conference on Advances in Recent Technologies in Communication and Computing. IEEE Computer Society.

[16] D. R. Lutz and D. N. Jayasimha, "The Power of Carry Save Addition," Department of Computer and Information Science, The Ohio State University. 1994.

[17] M. A. Franklin and T. Pan, "Performance Comparison of Asynchronous Adders," Proceedings of IEEE Symposium on Advanced Research in Asynchronous Circuits and Systems, Salt Lake City, 3-5 November 1994, 117-125. https://doi.org/10.1109/ASYNC.1994.656299

[18] J.L. Hennessy and D. A. Patterson "Computer Architecture: A Quantitative Approach," Morgan Kaufmann, Waltham. 1990.

[19] J. H. Kang, H. Shin, K.S. Kim, et al. Monolithic 3D integration of 2D materials-based electronics towards ultimate edge computing solutions. Nat. Mater. 22, 1470–1477 (2023). https://doi.org/10.1038/s41563-023-01704-z

[20] C. Wang, G. Shi, F. Qiao, R. Lin, S. Wu and Z. Hu, "Research Progress in Architecture and Application of RRAM with Computing-In-Memory," Nanoscale Adv., 2023, 5, 1559-1573.

[21] M. Abrar, H. Elahi, B. A. Ahmad et al, "An area-optimized N-bit multiplication technique using N/2-bit multiplication algorithm," SN Appl. Sci. 1, 1348 (2019). https://doi.org/10.1007/s42452-019-1367-6

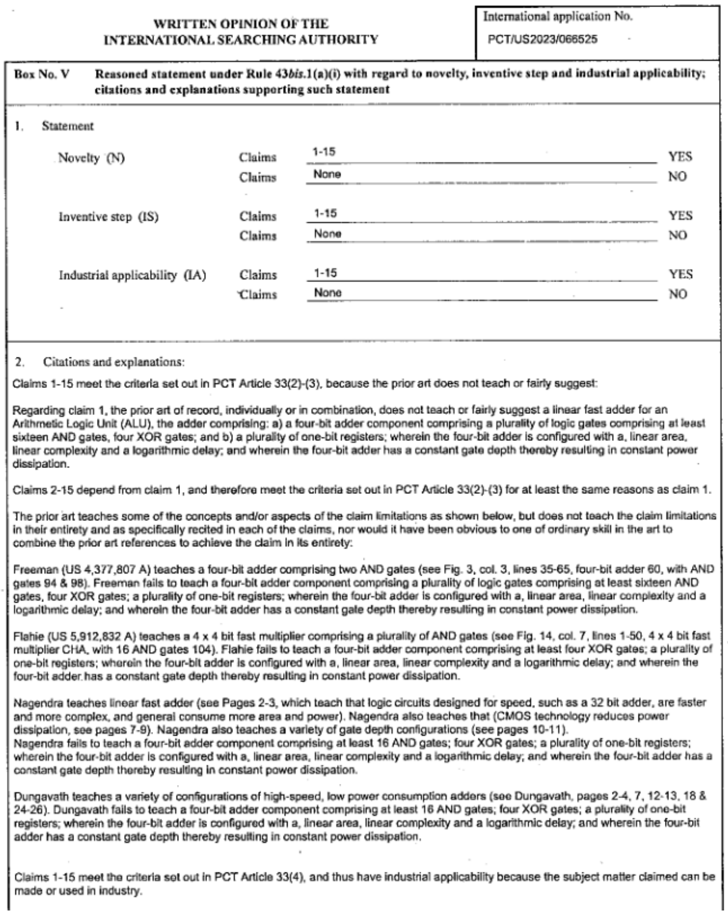
[22] N. Emmart and C. C. Weems, "High Precision Integer Multiplication with a GPU Using Strassen's Algorithm with Multiple FFT Sizes," Parallel Processing Letters, Vol.21, No. 03, pp. 359-375 (2011). https://doi.org/10.1142/S0129626411000266

[23] M. I. M. Taib, M. N. Z. Nazri, et. al., "Design of Multiplication and Division Operation for 16 Bit Arithmetic Logic Unit (ALU)," JOURNAL OF ELECTRONIC VOLTAGE AND APPLICATION VOL. 1 NO. 2 (2020), 46-54. DOI: https://doi.org/10.30880/jeva.2020.01.02.006

[24] M. Z. Rahman "Parallel Self-Timer Adder (PASTA)," United States Patent Application, May 9, 2013.

[25] T. Zhang, C. Xu, T. Li, Y. Qin, M. Nie, "An Optimized Floating-Point Matrix Multiplication on FPGA," Information Technology Journal, 12: 2013 1832-1838. DOI: 10.3923/itj.2013.1832.1838

VII. Appendix A



WRITTEN OPINION OF THE INTERNATIONAL SEARCHING AUTHORITY

International application No. PCT/US2023/066525

Box No. V Reasoned statement under Rule 43*bis*.1(a)(i) with regard to novelty, inventive step and industrial applicability; citations and explanations supporting such statement

1. Statement

Novelty (N)	Claims	1-15	YES
	Claims	None	NO
Inventive step (IS)	Claims	1-15	YES
	Claims	None	NO
Industrial applicability (IA)	Claims	1-15	YES
	Claims	None	NO

2. Citations and explanations:

Claims 1-15 meet the criteria set out in PCT Article 33(2)-(3), because the prior art does not teach or fairly suggest:

Regarding claim 1, the prior art of record, individually or in combination, does not teach or fairly suggest a linear fast adder for an Arithmetic Logic Unit (ALU), the adder comprising: a) a four-bit adder component comprising a plurality of logic gates comprising at least sixteen AND gates, four XOR gates; and b) a plurality of one-bit registers; wherein the four-bit adder is configured with a, linear area, linear complexity and a logarithmic delay; and wherein the four-bit adder has a constant gate depth thereby resulting in constant power dissipation.

Claims 2-15 depend from claim 1, and therefore meet the criteria set out in PCT Article 33(2)-(3) for at least the same reasons as claim 1.

The prior art teaches some of the concepts and/or aspects of the claim limitations as shown below, but does not teach the claim limitations in their entirety and as specifically recited in each of the claims, nor would it have been obvious to one of ordinary skill in the art to combine the prior art references to achieve the claim in its entirety:

Freeman (US 4,377,807 A) teaches a four-bit adder comprising two AND gates (see Fig. 3, col. 3, lines 35-65, four-bit adder 60, with AND gates 94 & 98). Freeman fails to teach a four-bit adder component comprising a plurality of logic gates comprising at least sixteen AND gates, four XOR gates; a plurality of one-bit registers; wherein the four-bit adder is configured with a, linear area, linear complexity and a logarithmic delay; and wherein the four-bit adder has a constant gate depth thereby resulting in constant power dissipation.

Flahie (US 5,912,832 A) teaches a 4 x 4 bit fast multiplier comprising a plurality of AND gates (see Fig. 14, col. 7, lines 1-50, 4 x 4 bit fast multiplier CHA, with 16 AND gates 104). Flahie fails to teach a four-bit adder component comprising at least four XOR gates; a plurality of one-bit registers; wherein the four-bit adder is configured with a, linear area, linear complexity and a logarithmic delay; and wherein the four-bit adder has a constant gate depth thereby resulting in constant power dissipation.

Nagendra teaches linear fast adder (see Pages 2-3, which teach that logic circuits designed for speed, such as a 32 bit adder, are faster and more complex, and general consume more area and power). Nagendra also teaches that (CMOS technology reduces power dissipation, see pages 7-9). Nagendra also teaches a variety of gate depth configurations (see pages 10-11).
Nagendra fails to teach a four-bit adder component comprising at least 16 AND gates; four XOR gates; a plurality of one-bit registers; wherein the four-bit adder is configured with a, linear area, linear complexity and a logarithmic delay; and wherein the four-bit adder has a constant gate depth thereby resulting in constant power dissipation.

Dungavath teaches a variety of configurations of high-speed, low power consumption adders (see Dungavath, pages 2-4, 7, 12-13, 18 & 24-26). Dungavath fails to teach a four-bit adder component comprising at least 16 AND gates; four XOR gates; a plurality of one-bit registers; wherein the four-bit adder is configured with a, linear area, linear complexity and a logarithmic delay; and wherein the four-bit adder has a constant gate depth thereby resulting in constant power dissipation.

Claims 1-15 meet the criteria set out in PCT Article 33(4), and thus have industrial applicability because the subject matter claimed can be made or used in industry.

Fig. 6. Written opinion of the Internatinal Search Authority (ISA), regarding patentability of the SLFA.