On a Simple and Linear Fast Adder and Its Implementation for In-Memory Matrix Multiplication

Juan P. Ramirez
Jalisco, México
jramirez@binaryprojx.com;
jramirez@ddotslab.com;
Personal web-page: www.binaryprojx.com

***Abstract*—A Simple and Linear Fast Adder has been proposed as basis for a rectangular arrangement of Half Adders and memory elements, of size $n \times b$, that functions as an adder of n-many b-bit inputs. Gate depth and topological complexity are constant and minimum. The one-to-one correspondence between memory elements and arithmetic logic gates allow a two-level architecture. The levels are simply connected in a one-to-one manner, eliminating the Von Neumann bottleneck.**

***Index Terms*—Demo Track - 4-page, Applied Mathematics, Compute-In-Memory, Fast Adder, Matrix Multiplication**

I. Introduction

The Patent-Pending Simple and Linear Fast Adder is expected to outperform other parallel fast adders in terms of energy efficiency, time-delays, design and production costs because of its uniform and scaleable design. The international patent application is found at the WIPO on-line portal using ID/Number: WO/2023/220537 for conducting a PATENTSCOPE Simple Search. The linear adder is generalized into a rectangular circuit that 1) Adds multiple inputs and 2) Multiplies two inputs. It is obtained by connecting several copies of the SLFA, in a simple manner. The scaleability of this circuit allows for an area, energy and time efficient implementation of matrix multiplication, using the theoretical minimum circuit area and complexity. Compounded with these benefits, the circuit is implemented as a Compute-In-Memory Architecture. The justification and deduction of the addition and multiplication performed by this architecture [1] is included as Supplementary Material. An extensive preprint is available at the author's homepage, [2]. The set theory for these operations is proposed as a canonical description of mathematics. Applications to group theory, number theory, combinatorics, real analysis, among others, are included. The set theoretic basis is also found in [3].

II. Simple and Linear Fast Adder

The SLFA is defined to perform addition of two inputs in terms of a logarithmic-time Finite State Machine [1], [2]. An addition algorithm is described, that adds as sets, and not sequences. Each number is represented as a set of numbers, according to its Ackermann Coding. That is to say, the number $\sum_i 2^{x_i}$ is represented as the set $\{x_1, x_2, \dots, x_n\}$. To add two numbers 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 the two new sets is equal to the original addition $A \oplus B = A' \oplus B'$. The powers of 2 in $A \oplus B$ have only been rearranged. The term A' consisting of non-repeated powers (symmetric difference). The term B' is given in terms of the repeated powers. It is equal to $B' = s(A \cap B)$. In a finite number of iterations the intersection $A^{(k)} \cap B^{(k)} = \emptyset$ is equal 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 find the addition of $11 + 23 = 30$. The addition is the sum of sets $A \oplus B = \{0,1,3\} \oplus \{0,1,2,4\}$ because $11 = 2^0 + 2^1 + 2^3$ and $23 = 2^0 + 2^1 + 2^2 + 2^4$. The inputs are represented with two side-by-side columns.

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

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 these values need to be displaced one level up because the intersection is representing addition of two equal powers of 2 and $2^n + 2^n = 2^{n+1}$. This gives two new columns

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

Iterating again gives the columns

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

In the next iteration we get

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

If the process is iterated again it gives

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

If the process is iterated once more, it reaches a stable state

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

because the same two columns are given in the next iteration. Thus, the Finite State Machine has reached a stable state and the answer, $11 + 23 = 34$, is given in the left column. In general, the left column of state $S(t_{k+1})$ is given by the symmetric difference in state $S(t_k)$. The right column of state $S(t_{k+1})$ is given by a displacement, one level up, of the intersection in state $S(t_k)$. A stable state is reached in a finite number of steps when the right column is empty. The average number of steps for adding two n-bit numbers is $\max(n)$, and 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 symmetric difference in each bit is found with an XOR gate, while the intersection in the same bit is given by an AND gate. An n-bit SLFA consists of n subunits connected in series, each sub unit consisting of two bits of memory and one Half Adder (one XOR and one AND gate). This parallel adder of linear topology, and constant complexity and gate depth is shown in Fig. 1. Each XOR gate is connected to the left column register of the same significant bit, while the AND gate is connected to the right column register of the next significant bit. Each register must be a double-edge triggered register capable of reading on the rising edge and writing on the falling edge.

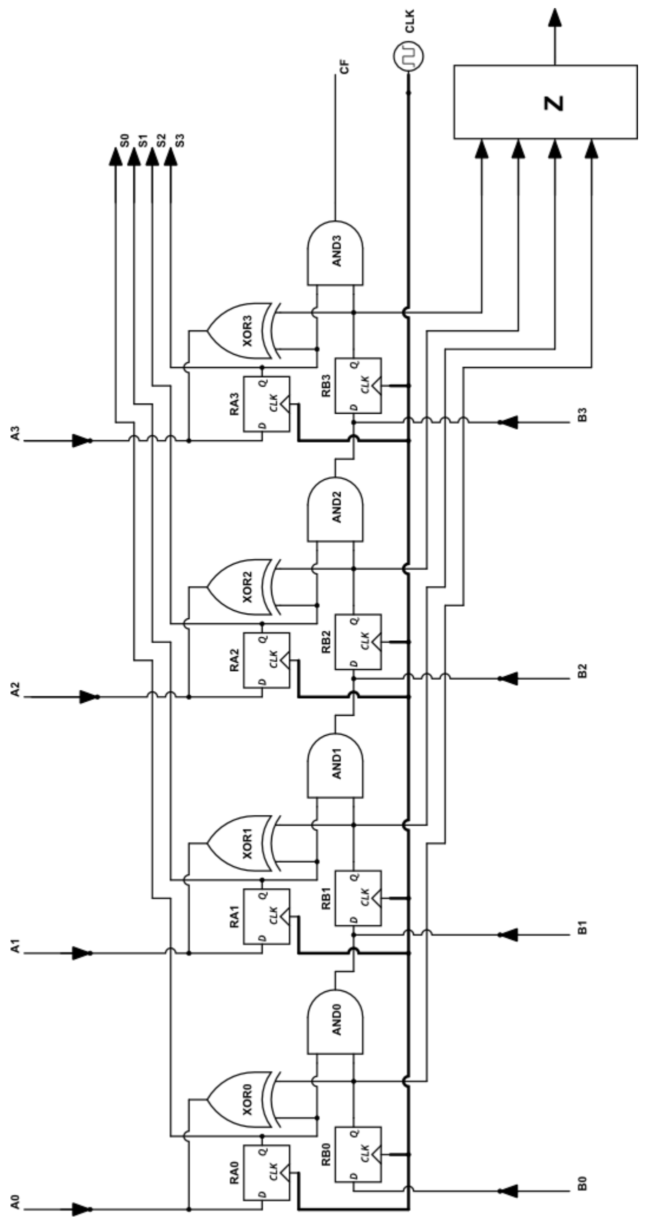

Fig. 1. A 4-bit SLFA, consisting of 4 subunits connected in series. Each subunit is made up of a half adder and a 2-bit register. The 'XOR' gates find the symmetric difference, while the intersection is found in the output of the 'AND' gates. A bit shift is applied to the intersection, and the process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

III. Generalizing the SLFA

A method is described for reducing the addition of n-many inputs to the addition of two inputs, which is a generalization of the Finite State Machine for adding two inputs. This column reduction algorithm is described in [1], [2], and we only briefly outline it here. The sum of k summands is reduced to $\max(k) + 1$ summands. Consider the sum of 4-many, 8-bit numbers. Let $A = a_0 a_1 \cdots a_7$, $B = b_0 b_1 \cdots b_7$,

$C = c_0 c_1 \cdots c_7$, $D = d_0 d_1 \cdots d_7$.

$$\begin{array}{cccc} a_7 & b_7 & c_7 & d_7 \\ a_6 & b_6 & c_6 & d_6 \\ a_5 & b_5 & c_5 & d_5 \\ a_4 & b_4 & c_4 & d_4 \\ a_3 & b_3 & c_3 & d_3 \\ a_2 & b_2 & c_2 & d_2 \\ a_1 & b_1 & c_1 & d_1 \\ a_0 & b_0 & c_0 & d_0, \end{array} \tag{1}$$

where each a_i, b_i, c_i, d_i takes a value in $\{0, 1\}$. There are a total of four columns. Therefore, three bits are sufficient for counting how many 1's are contained in a single row of (1) because $\max(4) + 1 = 2 + 1 = 3$. A 3-column grid with $8 + (3 - 1) = 10$ many rows will represent the same addition as the original four columns of (1).

$$\begin{array}{ccc} 0 & 0 & c'_9 \\ 0 & b'_8 & c'_8 \\ a'_7 & b'_7 & c'_7 \\ a'_6 & b'_6 & c'_6 \\ a'_5 & b'_5 & c'_5 \\ a'_4 & b'_4 & c'_4 \\ a'_3 & b'_3 & c'_3 \\ a'_2 & b'_2 & c'_2 \\ a'_1 & b'_1 & 0 \\ a'_0 & 0 & 0 \end{array} \tag{2}$$

The elements a'_0, b'_1, c'_2 are used to write the number of 1's in row 0 of (1). 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. The three column grid can be reduced to two columns, by the same process.

A rectangular version of the SLFA can be adapted to carry out this column reduction method for addition of n-many b-bit inputs. A rectangular grid of $n \times b$ nodes is formed by connecting multiple SLFAs side-by-side using simple connections between nodes, as shown in Fig. 2. Only nodes on the same row or column are connected.

Memory is placed on one level and Half Adders in a second level. Both levels are a simple rectangle circuit of equal size, and they are connected in one-to-one manner. Each memory node is connected to its corresponding HA, as in Fig. 3.

We can conveniently scale the circuit of Fig. 2 to find the dot product $\mathbf{u} \cdot \mathbf{v}$ of two n-vectors. A single copy of the rectangular circuit of Fig. 2 allows us to find one of the partial products of the dot product. Having n-many copies of the rectangular circuit, as shown below in Fig. 4, we can simultaneously calculate the partial products of $\mathbf{u} \cdot \mathbf{v}$. That is to say, each expression $u_i v_i$ is calculated in the i-th copy, for every $i = 1, 2, \ldots, n$.

The outputs of the n-many Multiple Input Adders of Fig. 4 can be sent to another Multiple Input Adder that will add the partial products $\sum_i u_i v_i$. Let $\mathbf{U}, \mathbf{V}$ two matrices of size $p \times n$ and $n \times q$, respectively. Taking q-many copies of the circuit M_1 in Fig. 4, we obtain a larger scale rectangular arrangement that calculates all the partial products for finding an entire row

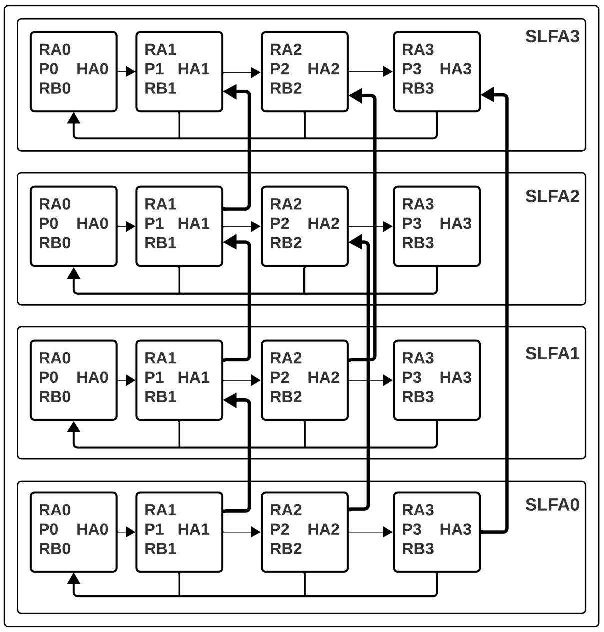

Fig. 2. A $b \times n$ grid constitutes an adder of n-many b-bit numbers that simulates the column reduction algorithm. Each row consists of an n-bit SLFA. Every node of each SLFA has an extra one-bit register, called the principal bit. Every SLFA conforming a row, counts the number of 1's in that row. Then, the results are sent to principal bits of the same column, in a diagonal manner that simulates the column reduction algorithm.

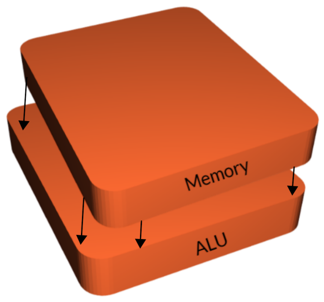

Fig. 3. The multiple input adder can be organized in two layers. One layer is reserved for memory registers, and the other layer is a grid of half adders.

of the product matrix $\mathbf{U} \times \mathbf{V}$, in parallel. A set of q-many Multiple Input Adders is used to add the respective partial products. Each one of these q-many adders finds an element for the given row of the product matrix. An entire row of the product matrix is found in parallel in the time that it takes to multiply two b-bit numbers (finding partial products), plus the time it takes to add n-many $2b$-bit inputs (adding partial products). This circuit is illustrated in Fig. 5.

IV. Conclusion

My mathematical research provides an optimal representation of all mathematical objects and structures [1], [2], [3], with the potential to revolutionize computer science and applied mathematics. Applications include a general method

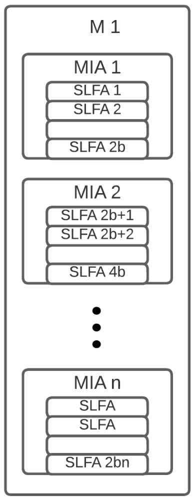

Fig. 4. M_1 consists of n-many Multiple Input Adders, each one able to compute a partial product of the dot product $\mathbf{u} \cdot \mathbf{v}$.

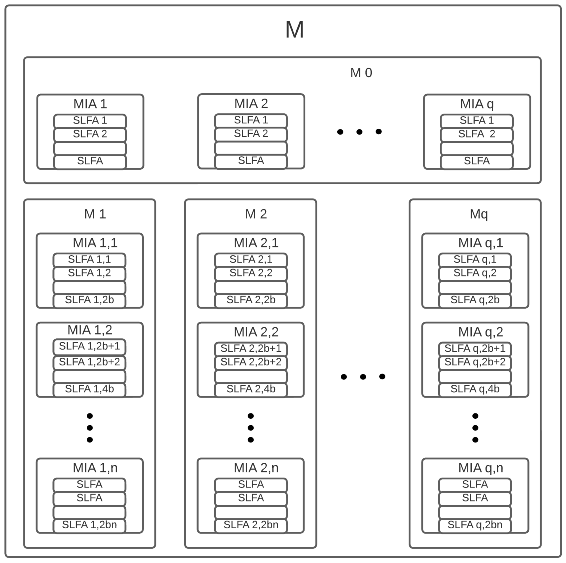

Fig. 5. Using a total of q-many units $M_1, \ldots, M_q$, it is possible to execute the partial products of a row. The multiple input adders of M_0 will add the corresponding partial products. The multiple input adder MIA_i, of M_0, will add the partial products $MIA_{(i,1)}, \ldots, MIA_{(i,n)}$, calculated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

for a new processor architecture with unique capabilities and characteristics that can replace the existing Von-Neumann Architecture with a Computing-In-Memory scheme, which is key in achieving time and energy efficient AI and Neural Network training

Through a fundamental solution to the problem of numerical representations and their computational complexity, we seek to transform the computing industry. This initiative is based on a major revision of mathematical foundations that allows fast and low-powered calculation of mathematical operations.

Mathematics is the invisible framework supporting the technology and innovations we encounter every day. Not only does it advance our understanding of abstract concepts, it also provides practical solutions to real-world challenges because it is the universal language behind all sciences. The coding language that computers understand is built on mathematics. Consequently, by refining the underlying principles of mathematics, we essentially upgrade the 'language' that enables our technology to communicate and perform tasks more efficiently. Using a novel conceptualization of the foundations of mathematics, with immediate applications in a number of key technologies facilitating seamless data representation and computational operations, we introduce arithmetic efficiency in ways previously unexplored at the Hardware level.

References

[1] "Simple and Linear Fast Adder of Multiple Inputs and Its Implementation in a Compute-In-Memory Architecture," Proc. of International Conference on Artificial Intelligence, Computer, Data Sciences and Applications. 2024, Victoria-Seychelles, 70.

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