LITERATURE REVIEW

Most modern-day computing systems, which rely on the von Neumann computing model, suffer from a problem known as the von Neumann bottleneck, where the separation of the control and memory units leads to restricted computing performance. This is a result of a computer's memory wall, the rate at which data is fetched and shuttled from the memory unit to the processor and back for processing and storage. In-memory computing, a promising and revolutionary computing paradigm, seeks to resolve this issue as the approach allows for memory retrieval, storage, and processing to occur in the same unit. Methods for allowing in-memory computing to take place have been organized in the past, an example being flow-based computing. However, these methods are reliant on the ability to synthesize nanoscale crossbars from Boolean formulas so that they can evaluate arbitrary programs. Our goal in this review is to propose a solution to efficiently construct such crossbar arrays and to analyze the concepts and literature surrounding our solution.

Exploring the Possibilities of "Boolean":

The term "Boolean logic" was coined by George Boole, an English mathematician who proposed a system of algebra that is based on Aristotle's propositional logic, in which all statements can be determined as either true or false. It is common for a wide variety of problems to be represented as boolean expressions. Boolean expressions can be evaluated, in much the same way as a mathematical expression, to produce a singular output of either true or false. In computing, the values "true" and "false" are represented by the binary values 1 and 0, respectively. A Boolean expression consists of one or more

input variables, which can also take on either a value of true or false, and Boolean operators, which combine variables in predefined ways. The Boolean AND operator requires every input variable to be true to have an output value of true while the Boolean OR operator only requires one input value of true. The Boolean NOT operator reverses the logical value of a Boolean expression, so if the input variable is true, then the output would be false and vice-versa. The best way to represent the behavior of such Boolean logic functions is through a truth table, which lists all possible values of input variables along with the corresponding output of the Boolean expression.

Memristors and Crossbars:

The memristor, a two-terminal device, is the most recent of the four fundamental circuit elements postulated in a 1971 paper by Leon Chua, a professor at the University of California, Berkeley at the time. For more reference, the original three ideal circuit elements are resistors, inductors, and capacitors. “Memristor” represents a contraction between the words “memory” and “resistor”, and the device itself relates magnetic flux and charge in a circuit. A memristor’s resistance depends on three factors, the magnitude, polarity, and time length of the voltage that has been applied. Once the voltage has been turned off, the memristor remembers its most recent resistance until it is turned on again, regardless of the time interval. A crossbar is a two-dimensional grid of wires which are connected by a memristive switch that lies at any point where a horizontal and a vertical wire cross. Likewise, crossbar architecture consists of an entire mesh of these fully interconnected wires and switches. It is important to note that these crossbars incorporate

newly developed memristive technologies leading to the creation of memristor-based nanoscale crossbar arrays.

Introducing Flow-based Computing:

A common problem in crossbar arrays is sneak paths, where the current can escape and sneak through other cells as it flows through unknown paths in parallel to the memristor. This can lead to inaccurate readings of memristor states. As a result, researchers have tried to develop sneak-path constraints to provide a solution to this phenomenon. Flow-based computing is an emerging in-memory computing paradigm that instead exploits the existence of sneak paths to perform computation, serving as the hardware structure where in-memory computing takes place. Flow-based computing on nanoscale memristor crossbars can be used for the evaluation of Boolean expressions. For this to happen, a pattern of loading data onto a two-dimensional array of nanoscale memristors must be designed so that the current flow through the crossbars can perform the desired computation. For example, if we were to take a simple Boolean formula such as “a AND b”, we would first turn off all memristors and load data corresponding to the values of the Boolean variables onto the memristors. This is possible because Boolean formulas have already been mapped to the crossbar design as each memristor assumes the value of a Boolean literal, “a” or “b”. From there, we decide to switch certain memristors “on” and others “off” depending on the pattern of data loading. A flow of current is then introduced in a specific input nanowire and reaches the output nanowire only if the Boolean formula evaluates to true which can be determined by checking if the current

flow is observable at the predefined wire. However, if the expression evaluates to false, the current flow will not be observable.

To construct memristor-based nanoscale crossbars more efficiently than before, we propose to automate the synthesis of crossbars to evaluate arbitrary programs; fixed-precision arithmetic is the subset of arbitrary programs that we plan to experiment around with. To accomplish this, we plan to implement a random search algorithm that can randomly select crossbar designs from a set of all possible designs and verify if they are equivalent to a target Boolean expression. The selected designs will have certain constraints on the crossbar dimensions and the number of variables. Since we cannot conduct our experiments physically due to infeasibility and high costs, it is important to note that our crossbar designs are synthesized digitally. In essence, they can be thought of as virtual blueprints. Finally, we will use heuristics to guide our search algorithm since it takes a very long time to go through all of the designs. However, one potential drawback of our proposed solution is the fact that the overall complexity of our operations is reduced since our analysis is limited to arithmetic only.

Works Cited

Chakraborty, D., Raj, S., & Jha, S. K. (n.d.). *Input-Aware Flow-Based Computing on Memristor Crossbars With Applications to Edge Detection*. Retrieved April 10, 2022, from https://par.nsf.gov/servlets/purl/10192475

Computation of Boolean Formulas Using Sneak Paths in Crossbar Computing ... - eecs.ucf.edu. (n.d.). Retrieved April 10, 2022, from http://eecs.ucf.edu/~velasquez/HIM.pdf

Introduction to Boolean Logic. GeeksforGeeks. (2022, January 27). Retrieved April 10, 2022, from https://www.geeksforgeeks.org/introduction-to-boolean-logic/

Williams, R. S. (2022, March 9). *How We Found the Missing Memristor*. IEEE Spectrum. Retrieved April 10, 2022, from https://spectrum.ieee.org/how-we-found-the-missing-memristor