**Stochastic Computing-Based Energy Aware (SEA) Accelerators**

Github Link: <https://github.com/hplp/Stochastic-PIM>

**Introduction**

Rahul does technical details?

**Duty Cycle Calculator**

The most important algorithm throughout the process of development was the duty cycle finder for all pulses. Using the duty cycle, all stages of operation (convolution, activation, and pooling) can be verified by comparing the program output to theorical values that were pre-determined. Additionally, using the duty cycles allowed for more efficient comparison of outputs between the C++ algorithm and MATLAB script because the vectors generated were at a length of 2 million. An additional helper function was made to find the average duty cycle over all pulses.

To find the duty cycle, the pulse is scanned until a rising edge is found and that value is stored. Then, the falling edge is found and stored. Finally, the rising edge that comes after the falling edge is found. Once all three values have been found, the ratio between the high pulse and low pulse can be calculated. This is a modified version of MATLAB’s duty cycle function that uses midpoints instead of rising and falling edges.

**Asynchronous Sigma Delta Modulator and Stochastic Computing Adder**

Prior to the development of this project, a behavioral model of the Asynchronous Sigma Delta Modulator (ASDM) and Asynchronous Stochastic Computing Adder (ASC adder) were created in MATLAB. The first step in developing the architecture was to convert the MATLAB script to C++. Afterwards, the test benches were made in C++ based off the ones modeled in MATLAB. The ASDM and ASC adder were verified by comparing the duty cycles generated within C++ to the MATLAB script.

**Convolution Function**

Once the ASDM and ASC adder were finished, the convolution function was the first stage that could be modeled. At first, a small-scale model was made using only 1 column with 4 rows as a proof of concept. A modified version of Neurosim’s memory array was originally used. The modifications were to remove unneeded dependencies and change the read behavior of a cell to return conductance instead of current. A new memory cell object was eventually made for the following reasons: the actual amount of the Neurosim code was very minor, a new cell written specifically for this project would be easier to modify for future use, and finally this would allow run-time variables to be more easily accessible.

**Parser**

Since the activation and pooling function depended on the output of the convolution function, improving the convolution function was the next priority after an initial testbench was set up. The parser would allow for csv or txt files to be read instead of having to manually load in weight and input values. Two functions were made for the separate files, one for reading the weight values and one for reading the input features.

**Activation Function**

The activation function was relatively simple. It utilized the duty cycle algorithm to generate the correct output.

**Pooling Function**

No new code needed to be written for the pooling function since the ASC adder had all of the functionality needed.

**Entire System**

After the individual parts were completed, they were combined together. A significant difference between the StochasticPIM class and the testbenches is scalability. With the improved parser, the architecture can scale up based on reading input without having to specify dimensions. Within the StochasticPIM class, the activation, convolution, and pooling function were internally separated to analyze the intermediate values during data flow. Several optional breakpoints were also added to improve system testability. A make file was made to run either only the convolution function or all three steps at the same time, with the user option to output to a CSV file.