CE 6303.001

HW5 Report

**Luke Allen** NetID: laa170930

**Yu Feng** NetID: yxf160330

09/28/2021

# Problem Description:

The problem we were tasked with for Homework 5 was to implement the more efficient convolution procedure described in the MSDAP paper with only plus and shift operations. The equation we will be implementing is as follows:

This new convolution requires a coefficient and data input, similar to the previous homework, however it also has a new input Rj. Rj determines the number of terms necessary to calculate uj. Additionally, the coefficients are formatted differently than in Homework 4, as they are now 9-bit signed numbers, instead of having 8 bits for value and 8 bits for sign. The output of the program will be a 40-bit hexadecimal number and will represent the processed data.

# Algorithm Implementation:

## Input:

The input functions needed to be modified from Homework 4 to suit the requirements for Homework 5. Not only have we added an additional input file with this assignment, but we now also had to deal with the coefficient input file being variable length. In the previous assignment, the number of coefficients is fixed, however, now the number of coefficients is dynamic. To deal with this, we created a fileSize() function which determines the number of lines in the given file by counting the number of “\n” characters. This tells us the number of coefficients in the input file. With the required memory space information on hand, we used calloc() to dynamically allocate the memory space to store all coefficients on-the-fly. On the other hand, the data and Rj input files were both fixed in length, so static arrays were created for each of the input files. With the corresponding memory space prepared, the values from each of the files were read and input into their respective memory space or arrays using our readFile() function.

## Uj Calculation:

To calculate Uj, calculateUj() function is created. It uses startPoint and endPoint values passed as argument to determine which coefficients are used in coefficient memory space to calculate the current Uj. startPoint is determined by the value of previous Rj (prevRj = 0 when j = 0), and updates to endPoint value at the end of current Uj calculation; endpoint denotes the end of coefficient index for current Uj calculation, obtained by Rj[j] + prevRj. This method works since Rj is used to define the amount of coefficient terms used in Uj calculation, giving away custom boundaries in the coefficient memory space.

### Coefficient Decoding:

After knowing which coefficients are to use in each Uj, we extract all coefficients within the boundary and decode them for dataX-related information, and the polarity (positive/negative) when used in Uj calculation.

To get the sign from the 9-bit signed format of the coefficients, we used a bit mask of 0x0100, ANDed it with the coefficient to get only the sign bit, and then shifted it to the right 8 bits. Thus, we will have a value of 1 if the coefficient is negative, or 0 if the coefficient is positive. To extract the coefficient value, we ANDed the coefficient with a bit mask of 0x00FF to get only the lower 8 bits, which contain the information required to calculate the index to pull from data array, stored as coeffValue.

The index for data array can then be calculated by dataIndex = n – coeffValue. If the result of dataIndex is negative, 0 is accumulated to the current Uj result. Otherwise, the corresponding input data is pulled from the data array.

### Input Data Processing

Since the input data is a 16-bit integer, all input data is first signed extended to 24 bits. bitExtender24() function helps pads the input data according to their sign.

To align with the 64-bit output (outputs are supposed to be 40 bits, but C program restriction requires 64-bit value to represent 40-bit number) when a selected term is used in calculation, the 24-bit processed data is shifted to left 40 bits to occupy the most significant bits of the number before adding to Uj. In the case of negative coefficient, reported by coefficient sign above, the processed data will perform a 2’s complement transformation before shifting 40 bits to the left. After the operation, the piece of data is accumulated with other terms in Uj.

## Y(n) calculation

With Uj terms calculated, Y(n) is calculated by accumulating all Uj results calculated above, and right shift them for 1 bit after each accumulation. At the end, the Y(n) overall result is shifted right 24 bits to preserve the first 40 MSB’s in output.

## Output:

The results of our computation are stored in our output array. From there, we have a writeFile() function which creates a new output file and loops through the array, writing the contents of each index into a new line of the file in the correct 40-bit hexadecimal number format.