The University of Texas at Dallas

ASIC Assignment-4

AXM220237, UXK220003

Adithya Prathipna Mullan Koni, Usman Khan
2-17-2024

C++ Code Implementation

Summary:

This C++ program performs convolution operations on given input data. It reads input from four files provided by the user: the input data file, the filter coefficients file, the filter lengths file, and the output file. After reading the input data and formatting it according to certain rules, it performs convolution using the provided filter coefficients and filter lengths. The resulting values are then written to the output file.

Algorithm Overview:

1. The program defines functions to read and process hexadecimal values from files.
2. It defines a function to compute the 2's complement of a number.
3. The main convolution function, filter, computes the convolution operation by iterating over the input data, applying the filter coefficients, and accumulating the result.
4. In the main function, it checks if the correct number of arguments are provided. If not, it displays usage instructions.
5. It reads the filenames provided by the user from command-line arguments.
6. It reads and formats the input data and filter coefficients from the respective files.
7. It performs convolution using the filter coefficients and lengths.
8. The resulting values are then written to the output file.

Overall, this program implements a basic convolution operation on input data using provided filter coefficients and lengths.

Functions Definitions:

Here's a brief explanation of each function in the provided code:

1. readHexadecimalValuesFromFile(const string& filename): This function reads hexadecimal values from a file and returns them as a vector of unsigned integers. It opens the specified file, reads each hexadecimal value, converts it to its decimal equivalent, and stores it in a vector. If the file cannot be opened, it prints an error message and returns an empty vector.
2. readAndFormatHexValues(const string& filename): This function reads hexadecimal values from a file and formats them according to certain rules. It reads each line from the file, extracts the hexadecimal values, formats them as required, and stores them in a vector of long integers. The formatting involves checking the most significant bit of each value and adding padding zeros accordingly. If the file cannot be opened, it prints an error message and returns an empty vector.

3. twos_comp(long int num): This function computes the two's complement of a given number. It performs bitwise negation followed by adding 1 to the result, effectively computing the two's complement. It then returns the result.
4. filter(long int x_arr[], int r_arr[], int h_arr[], int size, int rsize): This is the main convolution function. It takes in arrays representing the input data, filter coefficients, and filter lengths, along with their sizes. It iterates over the input data, applying the filter coefficients, and accumulating the result. It returns a dynamically allocated array containing the convolution result.

Each function performs a specific task related to reading, formatting, and processing data for the convolution operation.

Code

```
#include <iostream>
#include <sstream>
#include <string>
#include <fstream>
#include <vector>
#include <iomanip>
#include <cstdlib>
using namespace std;

//Function to read the values from the files and return them.
vector<unsigned int> readHexadecimalValuesFromFile(const string& filename)
{
    vector<unsigned int> hexValues;
    ifstream inputFile(filename.c_str());

    if (!inputFile.is_open())
    {
        cerr << "Error opening the file: " << filename << endl;
        return hexValues; // Return empty vector
    }

    string hexValue;
    while (inputFile >> hexValue)
    {
        // Convert hexadecimal string to integer and store it in the vector
        unsigned int decimalValue;
        stringstream ss;
        ss << hex << hexValue;
        ss >> decimalValue;
        hexValues.push_back(decimalValue);
    }
```

```
    inputFile.close();
    return hexValues;
}

//Function to read the values from the files and return them in required format.
vector<long int> readAndFormatHexValues(const string& filename)
{
    vector<long int> formattedHexValuesAsLong;
    ifstream file(filename.c_str());
    if (!file.is_open())
    {
        cerr << "Error opening file: " << filename << endl;
        return formattedHexValuesAsLong;
    }

    string line;
    while (getline(file, line))
    {
        istringstream iss(line);
        string hexValue;
        while (iss >> hexValue)
        {
            long int x;
            stringstream ss;
            ss << hex << hexValue;
            ss >> x;

            ostringstream formattedStream;
            if ((x & 0x8000) == 0x8000) {
                formattedStream << "FFFFFF" << uppercase << setfill('0') <<
setw(4) << hex << x << "0000";
            } else {
                formattedStream << "000000" << uppercase << setfill('0') <<
setw(4) << hex << x << "0000";
            }

            formattedHexValuesAsLong.push_back(strtol(formattedStream.str().c_str
(), NULL, 16));
        }
    }
}

//Function that returns 2's comp of any value.
```

```
long int twos_comp(long int num)
{
    num = ~num;
    return num +1;
}

// Main convolution function
long int* filter(long int x_arr[], int r_arr[], int h_arr[], int size, int rsize)
{
    int n,k;
    long int* y = new long int[size];
    long int temp1;
    for(n=0;n<size;n++)
    {
        int pos = 0,i,j;
        temp1 = 0;
        for(i = 0; i<rsize; i++)
        {
            long int u=0, temp;
            //Computation of each u term, i.e. u1, u2, u3...
            for(j=pos;j<(pos + r_arr[i]);j++)
            {
                int h = h_arr[j];
                int h1 = h_arr[j] & 0x0ff;
                temp =0;
                if((n-h1)<0)
                    continue;
                if((h>>8)&1 ==1)
                    temp = twos_comp(x_arr[n-h1]);
                else
                    temp = x_arr[n-h1];
                u+= temp;
            }
            //Accumulating u terms and shifting right
            pos+=r_arr[i];
            temp1+= u;
            temp1 = temp1 >>1;
        }
        y[n] = temp1;
    }
    return y;
}

int main(int argc, char *argv[])
```

```
{
    //From line 100 to 108 is about storing the files provided by the user
    if (argc != 5)
    {
        cerr << "Usage: " << argv[0] << " filename filenamecoeff filenameout" <<
std::endl;
        return 1;
    }

    string filename = argv[1];
    string filenamecoeff = argv[2];
    string filenameRj = argv[3];
    string filenameout = argv[4];


    long int* result_int;
    string result[1000];

    vector<long int> formattedHexValuesAsLong = readAndFormatHexValues(filename);
    long int x_arr[512];

    for (size_t i = 0; i < formattedHexValuesAsLong.size(); ++i)
    {
        x_arr[i] = formattedHexValuesAsLong[i];
    }

    vector<unsigned int> hvalues = readHexadecimalValuesFromFile(filenamecoeff);
    int h_arr[159];
    for (size_t j = 0; j < hvalues.size(); ++j)
    {
        h_arr[j] = hvalues[j];
    }

    vector<unsigned int> Rjvalues = readHexadecimalValuesFromFile(filenameRj);
    int r_arr[16];
    for (size_t k = 0; k < Rjvalues.size(); ++k)
    {
        r_arr[k] = Rjvalues[k];
    }

    int size = sizeof(x_arr)/sizeof(x_arr[0]);
    int rsize = sizeof(r_arr)/sizeof(r_arr[0]);
    int len;
    result_int = filter(x_arr,r_arr,h_arr,size, rsize);
```

```
    for(int i=0;i<size;i++)
    {
        string zeros ("");
        stringstream ss;
        ss<<uppercase<<hex<<result_int[i];
        result[i] = (ss.str());
        len = 16 - result[i].size();
        while(len > 0 )
        {
            zeros = zeros + "0";
            len--;
        }
        result[i]  = (zeros + result[i]).substr(6);
    }


    ofstream outputFile(filenameout.c_str());
    if (!outputFile.is_open())
    {
        cerr << "Error opening the file: " << filenameout << endl;
        return 1;
    }

    for (int i = 0; i < size; i++)
    {
        outputFile << result[i] << endl;
    }
    outputFile.close();

    return 0;
}
```