##### Содержание

* Цель работы - 2
* Выполнение работы - 4
* Вывод - 8
* Литература - 8
* Приложение 1. Исходный код программы для исходных данных, заданных в формате с плавающей точкой - 9
* Приложение 2. Исходный код программы для исходных данных, заданных в формате с фиксированной точкой - 13

Приложение 3. Графики - 16

1. Цель курсовой работы

Изучение влияния конечной разрядной сетки специализированного процессора при выполнении линейной фильтрации сигналов на точность формируемого результата.

Данное исследование проводится в три этапа. В первом необходимо разработать алгоритм и написать программу, реализующую алгоритм быстрого преобразования Хартли (БПХ), с использованием форматов чисел с плавающей точкой для исходных данных и конечных результатов. Во втором этапе нужно разработать алгоритм и написать программу, реализующую тот же алгоритм линейной фильтрации с использованием форматов чисел с фиксированной точкой для исходных данных и конечных результатов с учетом заданной разрядности:

* 12 разрядов - разрядность отсчетов исходных данных
* 8 разрядов - разрядность промежуточных результатов
* 16 разрядов - разрядность конечных результатов

Далее результаты, полученные в программе, разработанной на первом этапе, используем как эталонные и на их основе выполняем следующее:

* строим зависимость среднеквадратичной погрешности от длины обрабатываемого вектора данных
* строим зависимости СКО от длины обрабатываемого вектора при:

а) увеличении разрядности исходных данных на 2 и 4 бита при той же разрядности результатов

б) уменьшении разрядности исходных данных на 2 и 4 бита при той же разрядности результатов

Согласно варианту, длина входного вектора меняется от 8 до 1024.

Пусть имеется специализированный процессор, реализующий заданную процедуру линейной фильтрации сигналов. Данный процессор предназначен для обработки данных в формате с фиксированной точкой при конечной длине разрядной сетки. При этом исходные данные (дискретные отсчеты сигнала) являются целыми числами со знаком разрядности 8 двоичных разрядов. Разрядность промежуточных и конечных результатов - 16 двоичных разрядов.

1. Выполнение работы

Алгоритм быстрого преобразования Хартли по своей сути схож с другими подобными методами преобразования, например, преобразованиями Фурье. Суть методов заключается в преобразовании исходных отсчетов сигнала в некоторый результирующий вектор отсчетов, при этом остается возможным обратная операция преобразования. Так, имея исходный сигнал, мы можем разложить его на составляющие сигнал гармоники, и проводить с ним все необходимые операции: фильтрация (отбрасывание ненужных гармоник), передача и обратное преобразование сигнала (при потере нескольких гармоник сигнал все равно можно восстановить, хотя и с потерей качества. Особенно актуально при передаче мультимедиа).

Реализация алгоритма схожа с быстрым преобразованием Фурье. Предположим, мы хотим определить преобразование Хартли для последовательности чисел с плавающей точкой при n = 0, 1, 2...N-1 где N = 2^i (i - целое число).

Пусть:

Преобразуются последовательности чисел с плавающей точкой , каждая длиной .

Конечной задачей является преобразование

Расширение выражения:

Далее это выражение может быть расширено до:

Из этого следует:

Последние два выражения образуют основу алгоритма быстрого преобразования Хартли.

Реализация описанного выше алгоритма приведена в Приложении 1.

На следующем этапе работы необходимо реализовать ранее описанный алгоритм, но при этом использовать форматы данных с фиксированной точкой.

Для реализации алгоритма необходимо организовать работу с типом данных для хранения чисел в формате с фиксированной точкой заданной разрядности. Далее необходимо реализовать все математические операции, а так же математические функции, такие как косинус и синус.

Полученная реализация метода transform представлена в Приложении 2. Входной тип данных - массив байтов, тип результата - массив чисел размером 2 байта.

Заключительный этап. Взяв за эталон результаты выполнения первой программы, построим зависимость среднеквадратичной погрешности от длины N обрабатываемого вектора данных:

Проанализируем влияние разрядности на точность результата. Для этого проведем четыре эксперимента, уменьшая и увеличивая разрядность входных данных на 2 и на 4.

График зависимости представлен на рисунке 3.

Далее попробуем увеличивать разрядность:

Увеличим на 2 бита. Полученная зависимость представленна в таблице:

График зависимости представлен на рисунке 4.

Далее увеличим разрядность на 4 бита. Получим следующее соотношение:

График данной зависимости представлен на рисунке 5.

1. Вывод

При ограниченной разрядной сетке и использовании форматов чисел с фиксированной точкой при вычислениях возникает значительная погрешность, искажающая результаты обработки по сравнению с ожидаемыми.

1. Литература

F.Piccinin The fast Hartley transform as an alternative to the fast fourier transform // Surveillance research laboratory. South Australia. 1988.

Тропченко А.Ю., Тропченко А.А. Цифровая обработка сигналов. Методы предварительной обработки. // Учебное пособие по дисциплине "Теоретическая информатика" СПб: СПбГУ ИТМО, 2009.

Сергеев В.В., Усачев А.В. Преобразование Хартли в задачах цифровой обработки двумерных сигналов // Компьютерная оптика. - М.: МЦНТИ, 1992.

1. Приложение 1.

package ru.ifmo.cs.spt.fht;

public class FHT {

public static FHT instance;

public static FHT getInstance() {

if(instance == null) {

instance = new FHT();

}

return instance;

}

private float[] cosTable;

private float[] sinTable;

private int[] reversedBits;

private float[] tempArr;

private void initializeTables(int maxN) {

if (maxN > 0x40000000)

throw new RuntimeException("N can't be more 2^30");

makeSinCosTables(maxN);

makeBitReverseTable(maxN);

tempArr = new float[maxN];

}

private void bitReverse(float[] x) {

for (int i = 0; i < x.length; i++)

tempArr[i] = x[reversedBits[i]];

for (int i = 0; i < x.length; i++)

x[i] = tempArr[i];

}

private void makeSinCosTables(int maxN) {

int n = maxN / 4;

cosTable = new float[n];

sinTable = new float[n];

double theta = 0.0;

double dTheta = 2.0 \* Math.PI / maxN;

for (int i = 0; i < n; i++) {

cosTable[i] = (float) Math.cos(theta);

sinTable[i] = (float) Math.sin(theta);

theta += dTheta;

}

}

private void makeBitReverseTable(int maxN) {

reversedBits = new int[maxN];

int nLog2 = log2(maxN);

for (int i = 0; i < maxN; i++)

reversedBits[i] = bitRevX(i, nLog2);

}

private int bitRevX(int x, int bitlen) {

int temp = 0;

for (int i = 0; i <= bitlen; i++)

if ((x & (1 << i)) != 0)

temp |= (1 << (bitlen - i - 1));

return temp;

}

private int log2(int x) {

int count = 31;

while (!isBitSet(x, count))

count--;

return count;

}

private boolean isBitSet(int x, int bit) {

return ((x & (1 << bit)) != 0);

}

public void transform(float[] x) {

int gpSize, numGps, Nlog2;

int bfNum, numBfs;

int Ad0, Ad1, Ad2, Ad3, Ad4, CSAd;

float rt1, rt2, rt3, rt4;

final int maxN = x.length;

if (sinTable == null)

initializeTables(maxN);

Nlog2 = log2(maxN);

bitReverse(x); // bitReverse the input array

gpSize = 2; // first & second stages - do

// radix 4 butterflies once thru

numGps = maxN / 4;

for (int gpNum = 0; gpNum < numGps; gpNum++) {

Ad1 = gpNum \* 4;

Ad2 = Ad1 + 1;

Ad3 = Ad1 + gpSize;

Ad4 = Ad2 + gpSize;

rt1 = x[Ad1] + x[Ad2]; // a + b

rt2 = x[Ad1] - x[Ad2]; // a - b

rt3 = x[Ad3] + x[Ad4]; // c + d

rt4 = x[Ad3] - x[Ad4]; // c - d

x[Ad1] = rt1 + rt3; // a + b + (c + d)

x[Ad2] = rt2 + rt4; // a - b + (c - d)

x[Ad3] = rt1 - rt3; // a + b - (c + d)

x[Ad4] = rt2 - rt4; // a - b - (c - d)

}

if (Nlog2 > 2) {

// third + stages computed here

gpSize = 4;

numBfs = 2;

numGps = numGps / 2;

for (int stage = 2; stage < Nlog2; stage++) {

for (int gpNum = 0; gpNum < numGps; gpNum++) {

Ad0 = gpNum \* gpSize \* 2;

Ad1 = Ad0; // 1st butterfly is different

// mults needed

Ad2 = Ad1 + gpSize;

Ad3 = Ad1 + gpSize / 2;

Ad4 = Ad3 + gpSize;

rt1 = x[Ad1];

x[Ad1] = x[Ad1] + x[Ad2];

x[Ad2] = rt1 - x[Ad2];

rt1 = x[Ad3];

x[Ad3] = x[Ad3] + x[Ad4];

x[Ad4] = rt1 - x[Ad4];

for (bfNum = 1; bfNum < numBfs; bfNum++) {

Ad1 = bfNum + Ad0;

Ad2 = Ad1 + gpSize;

Ad3 = gpSize - bfNum + Ad0;

Ad4 = Ad3 + gpSize;

CSAd = bfNum \* numGps;

rt1=x[Ad2]\*cosTable[CSAd]+x[Ad4]\*sinTable[CSAd];

rt2=x[Ad4]\*cosTable[CSAd]-x[Ad2]\*sinTable[CSAd];

x[Ad2] = x[Ad1] - rt1;

x[Ad1] = x[Ad1] + rt1;

x[Ad4] = x[Ad3] + rt2;

x[Ad3] = x[Ad3] - rt2;

}

}

gpSize \*= 2;

numBfs \*= 2;

numGps = numGps / 2;

}

}

}

}

1. Приложение 2.

public short[] transform(byte[] input) {

short[] x = new short[input.length];

// конвертируем в типро промежуточного результата

for (int i = 0; i < input.length; i++) {

x[i] = (short) (input[i] << 4);

}

int Ad0, Ad1, Ad2, Ad3, Ad4, CSAd;

short rt1, rt2, rt3, rt4;

final int N = x.length;

if (S == null) {

initializeTables(N);

}

final int Nlog2 = log2(N);

reverseArrayBits(x, Nlog2);

int gpSize = 2;

int numGps = N / 4;

for (int gpNum = 0; gpNum < numGps; gpNum++) {

Ad1 = gpNum \* 4;

Ad2 = Ad1 + 1;

Ad3 = Ad1 + gpSize;

Ad4 = Ad2 + gpSize;

rt1 = (short) (x[Ad1] + x[Ad2]);

rt2 = (short) (x[Ad1] - x[Ad2]);

rt3 = (short) (x[Ad3] + x[Ad4]);

rt4 = (short) (x[Ad3] - x[Ad4]);

x[Ad1] = (short) (rt1 + rt3);

x[Ad2] = (short) (rt2 + rt4);

x[Ad3] = (short) (rt1 - rt3);

x[Ad4] = (short) (rt2 - rt4);

}

if (Nlog2 > 2) {

gpSize = 4;

int numBfs = 2;

numGps = numGps / 2;

for (int stage = 2; stage < Nlog2; stage++) {

for (int gpNum = 0; gpNum < numGps; gpNum++) {

Ad0 = gpNum \* gpSize \* 2;

Ad1 = Ad0; // 1st butterfly is different // mults needed

Ad2 = Ad1 + gpSize;

Ad3 = Ad1 + gpSize / 2;

Ad4 = Ad3 + gpSize;

rt1 = x[Ad1];

x[Ad1] = (short) (x[Ad1] + x[Ad2]);

x[Ad2] = (short) (rt1 - x[Ad2]);

rt1 = x[Ad3];

x[Ad3] = (short) (x[Ad3] + x[Ad4]);

x[Ad4] = (short) (rt1 - x[Ad4]);

for (int bfNum = 1; bfNum < numBfs; bfNum++) {

Ad1 = bfNum + Ad0;

Ad2 = Ad1 + gpSize;

Ad3 = gpSize - bfNum + Ad0;

Ad4 = Ad3 + gpSize;

CSAd = bfNum \* numGps;

rt1 = (short) (((x[Ad2] \* C[CSAd]) >> 8) + ((x[Ad4] \* S[CSAd]) >> 8));

rt2 = (short) (((x[Ad4] \* C[CSAd]) >> 8) - ((x[Ad2] \* S[CSAd]) >> 8));

x[Ad2] = (short) (x[Ad1] - rt1);

x[Ad1] = (short) (x[Ad1] + rt1);

x[Ad4] = (short) (x[Ad3] + rt2);

x[Ad3] = (short) (x[Ad3] - rt2);

} /\* end bfNum loop \*/

} /\* end gpNum loop \*/

gpSize \*= 2;

numBfs \*= 2;

numGps = numGps / 2;

} /\* end for all stages \*/

} /\* end if Nlog2 > 2 \*/

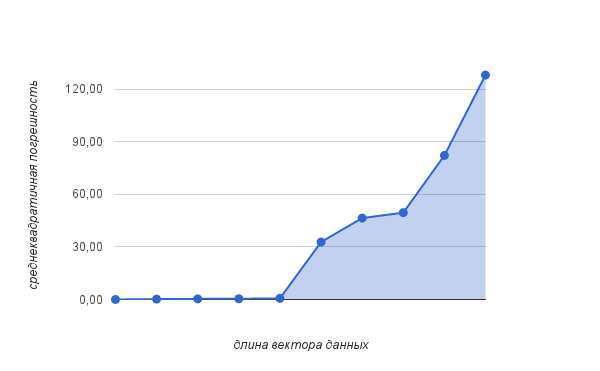
return x;

}

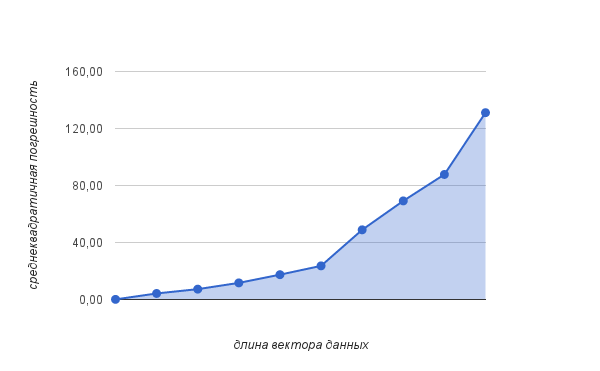
1. Приложение 3.



**Рис. 1.** График зависимости среднеквадратичной погрешности от длины обрабатываемого вектора данных



**Рис. 2.** График зависимости СКП от длины вектора входных данных при уменьшенной разрядности на 2 бита



**Рис. 3.** График зависимости СКП от длины вектора входных данных при уменьшенной разрядности на 4 бита



**Рис. 4.** График зависимости СКП от длины вектора входных данных при увеличенной разрядности на 4 бита



**Рис. 5.** График зависимости СКП от длины вектора входных данных при увеличенной разрядности на 2 бита