МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)**

Институт информационных технологий, математики и механики

Кафедра: Алгебры, геометрии и дискретной математики

Направление подготовки: «Фундаментальная информатика и информационные технологии»
Профиль подготовки: «Инженерия программного обеспечения»

ОТЧЕТ

по предмету «Анализ производительности и оптимизация программного обеспечения»

Выполнил:
студент группы
Кутовой В.Н.

Нижний Новгород
2020

Оглавление

Постановка задачи.

Преобразование Фурье— операция, сопоставляющая одной функции вещественной переменной другую функцию вещественной переменной. Эта новая функция описывает коэффициенты («амплитуды») при разложении исходной функции на элементарные составляющие — гармонические колебания с разными частотами (подобно тому, как музыкальный аккорд может быть выражен в виде суммы музыкальных звуков, которые его составляют).

Преобразование Фурье функции вещественной переменной является интегральным и задаётся следующей формулой:

$$\hat{f}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{\ ixw} \, dx.$$

Необходимо провести анализ производительности кода, реализующего вычисления прямого и обратного преобразования Фурье и провести процедуру его оптимизации.

Описание базовой версии кода и анализ производительности.

Базовая версия программы содержит реализацию прямого/обратного преобразования Фурье реализованные в функциях FDFT/RDFT, а также низкочастотный фильтр - функция LowPassFilter. (См. Приложение).

Для тестов было выбрано изображение размером 1200x1200 пикселей (рис.1).

Рис.1

Программа выполняет прямое преобразование Фурье, затем применяет низкочастотный фильтр и выполняет обратное преобразование Фурье восстанавливая изображение.

Характеристики тестовой машины:

- OS: Windows 10, 19041.685
- CPU: Intel Core i7-9700KF 3,6 GHz - 4,90 GHz; L1- 64K (per core), L2 - 256K (per core), L3 - 12MB (shared)
- RAM: 16Gb; 3200MHz; 16,18,18,38; Dual Channel; command rate 2T.

Проведем hotspot анализ производительности, используя Intel® VTune™ Profiler;

Elapsed Time: 200.834s

CPU Time:	197.808s
Total Thread Count:	10
Paused Time:	0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time
sinf	ucrtbase.dll	35.757s
FExp	MSVCP140.dll	33.587s
func@0x18002f3cf	MSVCP140.dll	32.507s
FDFT	DFT.exe	32.095s
RDFT	DFT.exe	31.553s
[Others]	N/A*	32.308s

*N/A is applied to non-summable metrics.

26	`for (int k = 0; k < cols; k++)`		
27	`for (int b = 0; b < rows; b++)`		
28	`{`		
29	`std::complex<float> sum(0.0, 0.0);`		
30	`for (int a = 0; a < cols; a++)`	0.1%	0.261s
31	`{`		
32	`int integers = -2 * k * a;`		
33	`std::complex<float> my_exponent(0.0, M_PI / cols * (float)integers);`	0.3%	0.681s
34	`sum += complexI.at<complex<float>>(a, b) * std::exp(my_exponent);`	9.7%	19.252s
35	`}`		
36	`fmatrix[k][b] = sum;`	0.0%	0.046s
37	`}`		

40	`for (int k = 0; k < cols; k++)`		
41	`for (int l = 0; l < rows; l++)`		
42	`{`		
43	`std::complex<float> sum(0.0, 0.0);`		
44	`for (int b = 0; b < rows; b++)`	0.2%	0.392s
45	`{`		
46	`int integers = -2 * l * b;`		
47	`std::complex<float> my_exponent(0.0, M_PI / rows * (float)integers);`	0.4%	0.737s
48	`sum += fmatrix[k][b] * std::exp(my_exponent);`	5.4%	10.663s
49	`}`		
50	`complexI.at<complex<float>>(k, l) = sum;`	0.0%	0.064s
51	`}`		

Довольно предсказуемо, что основное время занимает вычисление коэффициентов Фурье.

Запустим Memory Access анализ:

Elapsed Time: 26.530s

CPU Time:	189.921s	
Memory Bound:	**3.9%**	**of Pipeline Slots**
Loads:	562,434,672,534	
Stores:	137,009,510,162	
LLC Miss Count:	34,302,401	
Average Latency (cycles):	10	
Total Thread Count:	19	
Paused Time:	0s	

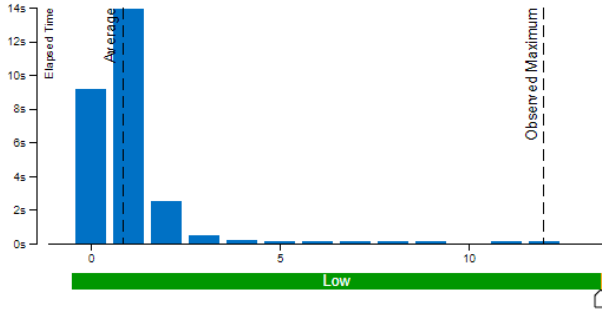
Bandwidth Utilization Histogram

Explore bandwidth utilization over time using the histogram and identify memory objects or fu

Bandwidth Domain: DRAM, GB/sec

Bandwidth Utilization Histogram

This histogram displays the wall time the bandwidth was utilized by certain value. Use sli view to group data and see all functions executed during a particular utilization type. To le provide maximum achievable DRAM and Interconnect bandwidth.



Как видно из анализа программа не сталкивается с ограничениями по пропускной способности памяти.

Проведем анализ параллельности кода:

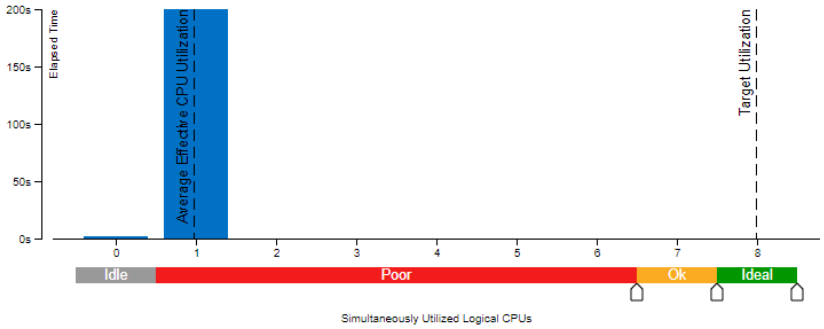
Elapsed Time: 200.713s

Paused Time: 0s

Effective CPU Utilization: 12.3% (0.984 out of 8 logical CPUs)

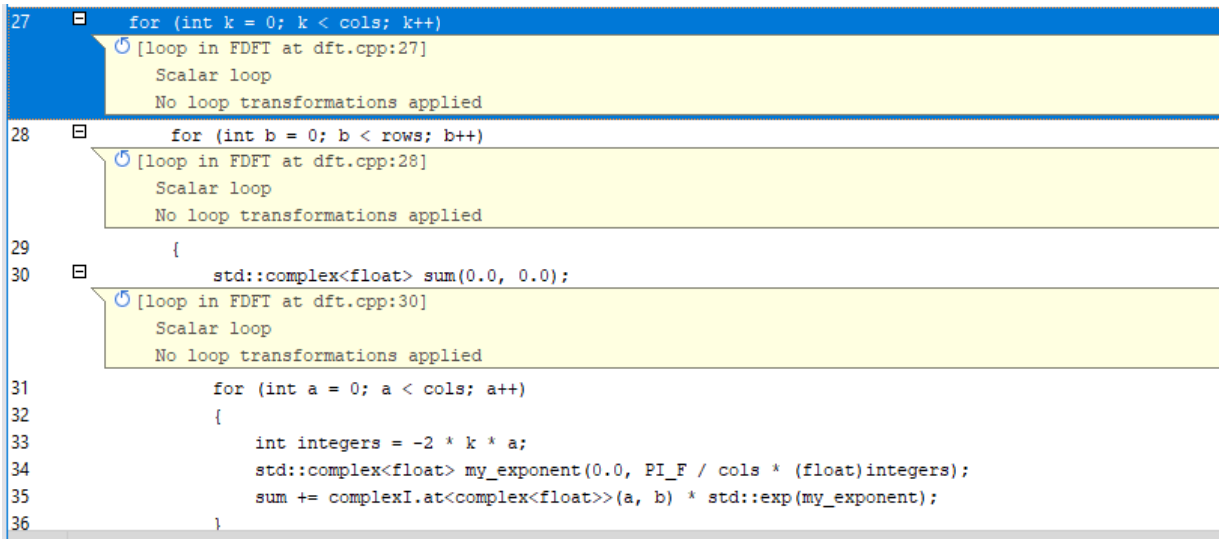
Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.



Программа выполняется последовательно, не используя все вычислительные ядра процессора.

Проведем анализ векторизации кода с помощью Intel® Advisor:



Из результатов видно, что представленные в коде циклы нельзя векторизовать из-за редукции в скаляр.

Процесс оптимизации.

Для решения проблемы неэффективного использования ядер процессора используем библиотеку OpenMP. Распараллелим основные циклы функций FDFT, RDFT, LowPassFilter и проведем повторный анализ с помощью Vtune:

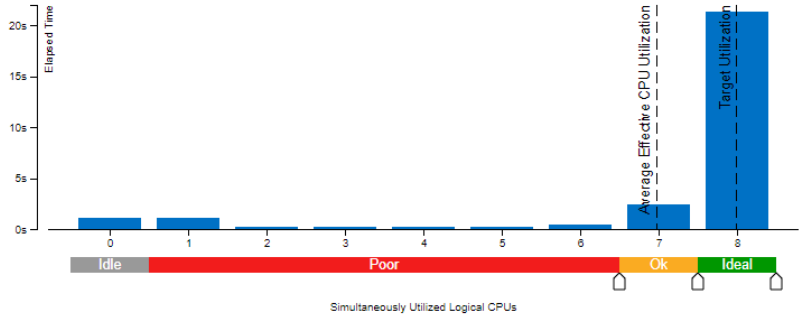
Elapsed Time: 27.171s

Paused Time: 0s

Effective CPU Utilization: 87.3% (6.981 out of 8 logical CPUs)

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.



Total Thread Count: 16

Thread Oversubscription: 0s (0.0% of CPU Time)

Wait Time with poor CPU Utilization: 16.220s (7.9% of Wait Time)

Top Waiting Objects

This section lists the objects that spent the most time waiting in your application. Objects can wait on specific calls, such as sleep() or I/O, or on contended synchronizations reflects high contention for that object and, thus, reduced parallelism.

Sync Object	Wait Time with poor CPU Utilization	(% from Object Wait Time)	Wait Count
Auto Reset Event 0xa84d7971	11.379s	6.4%	28
Manual Reset Event 0xa4ad42ca	3.066s	100.0%	24
Auto Reset Event 0x1c379a0d	1.631s	6.4%	4
Manual Reset Event 0x2bdd43e7	0.143s	100.0%	2
Read/Write Lock 0x5a231dff	0.000s	100.0%	8
[Others]	0.001s	100.0%	22

**N/A is applied to non-summable metrics.*

Заключение.

Проведя анализ производительности исходной программы, была выявлена проблема с неэффективным использованием ядер процессора. Используя технологию OpenMP нам удалось ускорить выполнение программы более чем в 7 раз.

Приложение. Код программы.

Исходный код программы доступен в репозитории https://github.com/VadimKutovoi/DFT

Базовая версия кода.

```
#define _USE_MATH_DEFINES
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
#include <stdint.h>
#include <complex>
#include <math.h>
#include <vector>

using namespace cv;
using namespace std;

Mat FDFT(Mat complexI)
{
	const unsigned int cols = complexI.cols;
	const unsigned int rows = complexI.rows;

	vector <vector<complex<float>>> fmatrix(rows);
	for (uint i = 0; i < rows; i++)
		fmatrix[i].resize(cols);

	for (uint k = 0; k < cols; k++)
		for (uint b = 0; b < rows; b++)
		{
			std::complex<float> sum(0.0, 0.0);
			for (uint a = 0; a < cols; a++)
			{
				int integers = -2 * k * a;
				std::complex<float> my_exponent(0.0, M_PI / cols * (float)integers);
				sum += complexI.at<complex<float>>(a, b) * std::exp(my_exponent);
			}
			fmatrix[k][b] = sum;
		}

	for (uint k = 0; k < cols; k++)
		for (uint l = 0; l < rows; l++)
		{
			std::complex<float> sum(0.0, 0.0);
			for (uint b = 0; b < rows; b++)
			{
				int integers = -2 * l * b;
				std::complex<float> my_exponent(0.0, M_PI / rows * (float)integers);
				sum += fmatrix[k][b] * std::exp(my_exponent);
			}
			complexI.at<complex<float>>(k, l) = sum;
		}
	return complexI;
}

Mat RDFT(Mat complexI)
{
	const unsigned int cols = complexI.cols;
	const unsigned int rows = complexI.rows;

	vector <vector<complex<float>>> fmatrix(rows);
	for (uint i = 0; i < rows; i++)
		fmatrix[i].resize(cols);
```

```
	for (uint k = 0; k < cols; k++)
		for (uint b = 0; b < rows; b++)
		{
			std::complex<float> sum(0.0, 0.0);
			for (uint a = 0; a < cols; a++)
			{
				int integers = 2 * k * a;
				std::complex<float> my_exponent(0.0, M_PI / cols * (float)integers);
				sum += complexI.at<complex<float>>(a, b) * std::exp(my_exponent);
			}
			fmatrix[k][b] = sum / (float)cols;
		}

	for (uint k = 0; k < cols; k++)
		for (uint l = 0; l < rows; l++)
		{
			std::complex<float> sum(0.0, 0.0);
			for (uint b = 0; b < rows; b++)
			{
				int integers = 2 * l * b;
				std::complex<float> my_exponent(0.0, M_PI / rows * (float)integers);
				sum += fmatrix[k][b] * std::exp(my_exponent);
			}
			complexI.at<complex<float>>(k, l) = sum / (float)rows;
		}
	return complexI;
}

Mat LowPassFilter(Mat complexI)
{
	int radius = min(complexI.cols, complexI.rows);
	Point2d center;
	center.x = int(complexI.cols / 2);
	center.y = int(complexI.rows / 2);

	for (uint x = 0; x < complexI.cols; x++)
		for (uint y = 0; y < complexI.rows; y++)
		{
			if ((x - center.x) * (x - center.x) + (y - center.y) * (y - center.y) < radius / 1.7 * radius / 1.7)
				complexI.at<complex<float>>(x, y) = complex<float>(0, 0);
		}
	return complexI;
}

void ShowMagnitude(Mat complexI, string windowname)
{
	Mat padded;
	int m = getOptimalDFTSize(complexI.rows);
	int n = getOptimalDFTSize(complexI.cols);
	copyMakeBorder(complexI, padded, 0, m - complexI.rows, 0, n - complexI.cols, BORDER_CONSTANT,
Scalar::all(0));

	Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };

	split(complexI, planes);
	magnitude(planes[0], planes[1], planes[0]);
	Mat magI = planes[0];

	magI += Scalar::all(1);
	log(magI, magI);

	magI = magI(Rect(0, 0, magI.cols & -2, magI.rows & -2));

	int cx = magI.cols / 2;
```

```
	int cy = magI.rows / 2;

	Mat q0(magI, Rect(0, 0, cx, cy));
	Mat q1(magI, Rect(cx, 0, cx, cy));
	Mat q2(magI, Rect(0, cy, cx, cy));
	Mat q3(magI, Rect(cx, cy, cx, cy));

	Mat tmp;
	q0.copyTo(tmp);
	q3.copyTo(q0);
	tmp.copyTo(q3);

	q1.copyTo(tmp);
	q2.copyTo(q1);
	tmp.copyTo(q2);

	normalize(magI, magI, 0, 1, CV_MINMAX);

	imshow(windowname, magI);
}

int main()
{
	const char* filename = "D:\\Documents\\Visual Studio
2015\\Projects\\ConsoleApplication1\\x64\\Release\\Lenna.jpg";

	Mat original = imread(filename, CV_LOAD_IMAGE_GRAYSCALE);
	int rsz = min(original.cols, original.rows);
	resize(original, original, Size(rsz, rsz), 0, 0, CV_INTER_LINEAR);
	imshow("Input Image", original);

	Mat padded;
	int m = getOptimalDFTSize(original.rows);
	int n = getOptimalDFTSize(original.cols);
	copyMakeBorder(original, padded, 0, m - original.rows, 0, n - original.cols, BORDER_CONSTANT, Scalar::all(0));

	Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };
	Mat complexI;

	merge(planes, 2, complexI);

	complexI = FDFT(complexI);

	ShowMagnitude(complexI, "Before low pass filter");

	complexI = LowPassFilter(complexI);

	ShowMagnitude(complexI, "After low pass filter");

	complexI = RDFT(complexI);

	split(complexI, planes);
	Mat magI = planes[0];

	normalize(magI, magI, 0, 1, CV_MINMAX);

	imshow("Low pass filtered", magI);

	waitKey();

	return 0;
}
```

Оптимизированная версия кода.

```
#define _USE_MATH_DEFINES
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/imgproc/imgproc_c.h"
#include "opencv2/highgui/highgui.hpp"
#include <opencv2/imgcodecs.hpp >
#include <iostream>
#include <stdint.h>
#include <complex>
#include <math.h>
#include <vector>

using namespace cv;
using namespace std;

Mat FDFT(Mat complexI)
{
	const unsigned int cols = complexI.cols;
	const unsigned int rows = complexI.rows;

	vector <vector<complex<float>>> fmatrix(rows);
	for (uint i = 0; i < rows; i++)
		fmatrix[i].resize(cols);

#pragma omp parallel for
	for (int k = 0; k < cols; k++)
		for (int b = 0; b < rows; b++)
		{
			std::complex<float> sum(0.0, 0.0);
			for (int a = 0; a < cols; a++)
			{
				int integers = -2 * k * a;
				std::complex<float> my_exponent(0.0, M_PI / cols * (float)integers);
				sum += complexI.at<complex<float>>(a, b) * std::exp(my_exponent);
			}
			fmatrix[k][b] = sum;
		}

#pragma omp parallel for
	for (int k = 0; k < cols; k++)
		for (int l = 0; l < rows; l++)
		{
			std::complex<float> sum(0.0, 0.0);
			for (int b = 0; b < rows; b++)
			{
				int integers = -2 * l * b;
				std::complex<float> my_exponent(0.0, M_PI / rows * (float)integers);
				sum += fmatrix[k][b] * std::exp(my_exponent);
			}
			complexI.at<complex<float>>(k, l) = sum;
		}
	return complexI;
}

Mat RDFT(Mat complexI)
{
	const unsigned int cols = complexI.cols;
	const unsigned int rows = complexI.rows;

	vector <vector<complex<float>>> fmatrix(rows);
	for (uint i = 0; i < rows; i++)
		fmatrix[i].resize(cols);

#pragma omp parallel for
	for (int k = 0; k < cols; k++)
```

```
		for (int b = 0; b < rows; b++)
		{
			std::complex<float> sum(0.0, 0.0);
			for (int a = 0; a < cols; a++)
			{
				int integers = 2 * k * a;
				std::complex<float> my_exponent(0.0, M_PI / cols * (float)integers);
				sum += complexI.at<complex<float>>(a, b) * std::exp(my_exponent);
			}
			fmatrix[k][b] = sum / (float)cols;
		}

#pragma omp parallel for
	for (int k = 0; k < cols; k++)
		for (int l = 0; l < rows; l++)
		{
			std::complex<float> sum(0.0, 0.0);
			for (int b = 0; b < rows; b++)
			{
				int integers = 2 * l * b;
				std::complex<float> my_exponent(0.0, M_PI / rows * (float)integers);
				sum += fmatrix[k][b] * std::exp(my_exponent);
			}
			complexI.at<complex<float>>(k, l) = sum / (float)rows;
		}
	return complexI;
}

Mat LowPassFilter(Mat complexI)
{
	int radius = min(complexI.cols, complexI.rows);
	Point2d center;
	center.x = int(complexI.cols / 2);
	center.y = int(complexI.rows / 2);

#pragma omp parallel for shared(complexI)
	for (int x = 0; x < complexI.cols; x++)
		for (int y = 0; y < complexI.rows; y++)
		{
			if ((x - center.x) * (x - center.x) + (y - center.y) * (y - center.y) < radius / 1.7 * radius / 1.7)
				complexI.at<complex<float>>(x, y) = complex<float>(0, 0);
		}
	return complexI;
}

void ShowMagnitude(Mat complexI, string windowname)
{
	Mat padded;
	int m = getOptimalDFTSize(complexI.rows);
	int n = getOptimalDFTSize(complexI.cols);
	copyMakeBorder(complexI, padded, 0, m - complexI.rows, 0, n - complexI.cols, BORDER_CONSTANT,
Scalar::all(0));

	Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };

	split(complexI, planes);
	magnitude(planes[0], planes[1], planes[0]);
	Mat magI = planes[0];

	magI += Scalar::all(1);
	log(magI, magI);

	magI = magI(Rect(0, 0, magI.cols & -2, magI.rows & -2));

	int cx = magI.cols / 2;
```

```
	int cy = magI.rows / 2;

	Mat q0(magI, Rect(0, 0, cx, cy));
	Mat q1(magI, Rect(cx, 0, cx, cy));
	Mat q2(magI, Rect(0, cy, cx, cy));
	Mat q3(magI, Rect(cx, cy, cx, cy));

	Mat tmp;
	q0.copyTo(tmp);
	q3.copyTo(q0);
	tmp.copyTo(q3);

	q1.copyTo(tmp);
	q2.copyTo(q1);
	tmp.copyTo(q2);

	normalize(magI, magI, 0, 1, CV_MINMAX);

	imshow(windowname, magI);
}

int main()
{
	const char* filename = "C:\\Users\\sirku\\Desktop\\Work\\lenna.png";

	Mat original = imread(filename, cv::ImreadModes::IMREAD_GRAYSCALE);
	int rsz = min(original.cols, original.rows);
	resize(original, original, Size(rsz, rsz), 0, 0, CV_INTER_LINEAR);
	imshow("Input Image", original);

	Mat padded;
	int m = getOptimalDFTSize(original.rows);
	int n = getOptimalDFTSize(original.cols);
	copyMakeBorder(original, padded, 0, m - original.rows, 0, n - original.cols, BORDER_CONSTANT, Scalar::all(0));

	Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };
	Mat complexI;

	merge(planes, 2, complexI);

	complexI = FDFT(complexI);

	ShowMagnitude(complexI, "Before low pass filter");

	complexI = LowPassFilter(complexI);

	ShowMagnitude(complexI, "After low pass filter");

	complexI = RDFT(complexI);

	split(complexI, planes);
	Mat magI = planes[0];

	normalize(magI, magI, 0, 1, CV_MINMAX);

	imshow("Low pass filtered", magI);
	waitKey();
	return 0;
}
```