МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Операционные системы

Отчет по лабораторной работе № 4

«Многопоточность»

Выполнил студент

группы М30-309Б-22:  
Татарян Г. В.  
Проверил:  
Титов Ю.П.

**Задание**

Многопоточный фильтр Собела. Программа получает фотографию на вход, делится на pthread  
 и накладывает фильтр Собела. Каждый поток обрабатывает определенное количество строк исходного изображения. Программа фиксирует время своего выполнения.

Запустить программу с 1,2,4,8, 16 и 32 потоками, зафиксировать время выполнения для каждой реализации. Сделать выводы.

Литература: Основы программирования в Linux. Автор: Мэтью Нейл, Стоунс Ричард. Глава 12. Потоки POSIX.

Варианты для всех студентов одинаковы, но желательно выполнять лабораторную работу на разных машинах.

**Код программы**

#include <opencv2/opencv.hpp>

#include <iostream>

#include <pthread.h>

#include <cmath>

#include <chrono>

using namespace std;

using namespace std::chrono;

using namespace cv;

// Собель X и Y ядра

const int kernelX[3][3] = {

{-1, 0, 1},

{-2, 0, 2},

{-1, 0, 1}

};

const int kernelY[3][3] = {

{1, 2, 1},

{0, 0, 0},

{-1, -2, -1}

};

// Исходное изображение и результат

Mat image;

Mat result;

// Структура данных для передачи параметров в потоки

struct ThreadData {

int startRow;

int endRow;

};

// Функция для применения фильтра Собела к части изображения

void\* applySobelFilter(void\* arg) {

ThreadData\* data = (ThreadData\*)arg;

int gx, gy;

for (int i = data->startRow; i < data->endRow; ++i) {

for (int j = 0; j < image.cols; ++j) {

gx = gy = 0;

// Apply Sobel kernels, with boundary checks

for (int k = -1; k <= 1; ++k) {

for (int l = -1; l <= 1; ++l) {

int row = i + k;

int col = j + l;

// Check boundaries

if (row >= 0 && row < image.rows && col >= 0 && col < image.cols) {

int pixel = image.at<uchar>(row, col);

gx += pixel \* kernelX[k + 1][l + 1];

gy += pixel \* kernelY[k + 1][l + 1];

}

}

}

int magnitude = sqrt(gx \* gx + gy \* gy);

result.at<uchar>(i, j) = saturate\_cast<uchar>(magnitude);

}

}

return nullptr;

}

int main() {

// Загрузка изображения в градациях серого

string imagePath = "/home/debian/Desktop/lr4/cheshire\_cat.jpg"; // Укажи путь к изображению

image = imread(imagePath, IMREAD\_GRAYSCALE);

if (image.empty()) {

cerr << "Ошибка загрузки изображения!" << endl;

return -1;

}

// Создаем матрицу для хранения результата

result = Mat::zeros(image.rows, image.cols, image.type());

// Количество потоков для тестирования

vector<int> threadCounts = {1, 2, 4, 8, 16, 32};

for (int threadCount : threadCounts) {

// Начало измерения времени

auto start = high\_resolution\_clock::now();

pthread\_t threads[threadCount];

ThreadData threadData[threadCount];

int rowsPerThread = image.rows / threadCount;

// Создаем потоки

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

threadData[i].startRow = i \* rowsPerThread;

threadData[i].endRow = (i == threadCount - 1) ? image.rows : (i + 1) \* rowsPerThread;

pthread\_create(&threads[i], nullptr, applySobelFilter, &threadData[i]);

}

// Ждем завершения всех потоков

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

pthread\_join(threads[i], nullptr);

}

// Конец измерения времени

auto end = high\_resolution\_clock::now();

auto duration = duration\_cast<milliseconds>(end - start).count();

cout << "Threads: " << threadCount << ", Time: " << duration << " ms" << endl;

}

// Сохранение результата

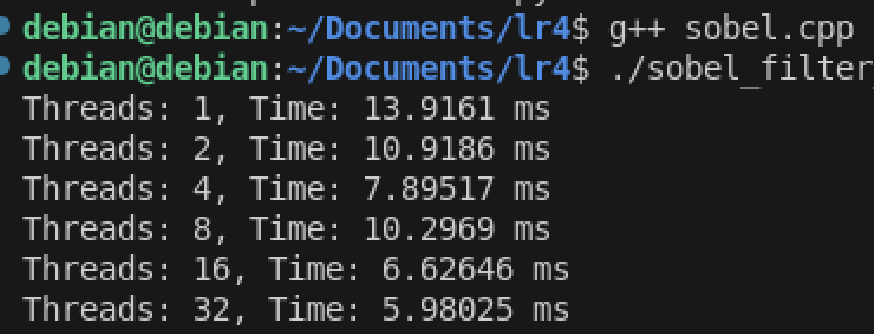
imwrite("sobel\_output.jpg", result);

return 0;

}

**Результаты**

Работа программы:



Картинка до и после обработки фильтром Собела

