САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛПЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

По домашней работе № 5

«OpenMP»

Выполнил: Рахмани Асаддулла Наджибуллаевич

Номер ИСУ: 334941

Студ.гр. – М3134

Санкт-Петербург

2021

Цель работы: знакомство со стандартом OpenMP.

Инструментарий и требования к работе: использование C++.

Теоретическая часть

OpenMP – набор директив компилятора, написанных на C++ и C, которые поддерживают параллельное программирование. При использовании OpenMP мы выделяем секции в коде, которые будут «распараллелены».

Секции в коде выделяются с помощью команды “#pragma omp parallel {}”, где фигурные скобки задают границы этой секции. Из наиболее часто встречающихся команд стоит отменить barrier и critical. Где barrier обозначает границу в секции, дальше которой «не идет» ни один тред, до тех пор, пока до нее не дойдут все остальные. A section обозначает такую часть кода в секции, что ее может выполнять только один тред за раз.

Также в openMP есть команда “#pramga omp for” – благодаря ей openMP сам распараллеливает цикл. Также для сокращения можно писать “#pragma omp parallel for”, не объявляя параллельную секцию в коде.

В дополнение к команде, что распараллеливает циклы можно дописать нечто подобное: “schedule(type, chanks)”, где type – чаще всего либо static, либо dynamic(есть еще другие типы, но они используются редко), a chanks – положительное целое число в пределах разумного. С помощью этой команды мы как бы говорим openMP каким образом мы хотим, чтобы он распараллеливал наш цикл.

Static означает, что цикл имеет тип статического «планирования», если так можно сказать. OpenMP делит итерации на фрагменты размером chank и распределяет их по тредам в циклическом порядке. Мы как бы уже знаем, как будет выполняться код на стадии компиляции.

Dynamic же значит, что цикл имеет тип «динамического» планирования. OpenMP также делит итерации на блоки размером chank, но каждый тред выполняет только часть итераций, а затем запрашивает другой фрагмент, пока они не закончатся. Суть в том, что мы не знаем какой будет порядок прохода по этим «фрагментам», OpenMP решает это во время работы кода.

Перед параллельными секциями в «последовательной» части кода стоит указать количество тредов, которое мы хотим, чтобы выделялось каждой секции. Это можно сделать с помощью команды “omp_set_num_threads(x)”. Чтобы получить количество тредов в текущей секции(а оно не всегда соответствует количеству тредов, что мы указали в предыдущей команде) можно воспользоваться командой “omp_get_num_threads()”.

Еще стоит отметить функцию “omp_get_wtime()”, с ее помощью я определял время работы программы.

Описание реализуемого алгоритма

Получив название файла и убедившись, что он существует и имеет нужное нам расширение, я считываю заголовок файла и его содержимое. Сами пиксели я храню, как последовательность байт.

После чего я нахожу максимальный и минимальный элемент в массиве, с учетом коэффициента, который подается на вход. Минимумом конкретного пикселя является минимум из его цветовых каналов, в случае если это «ppm». При поиске максимума я поступаю аналогично.

После чего я нахожу новое значение пикселя, благодаря формуле, что я вывел, куда подается значение старого пикселя, минимум и максимум, и перезаписываю его.

После чего я записываю в файл получившееся изображение.

Графики времени работы программы

Здесь каждое время работы = среднее арифметическое от 5 запусков программы при конкретной конфигурации.

Графики работы программы при schedule type = static и chanks = {1, 2, 4}:

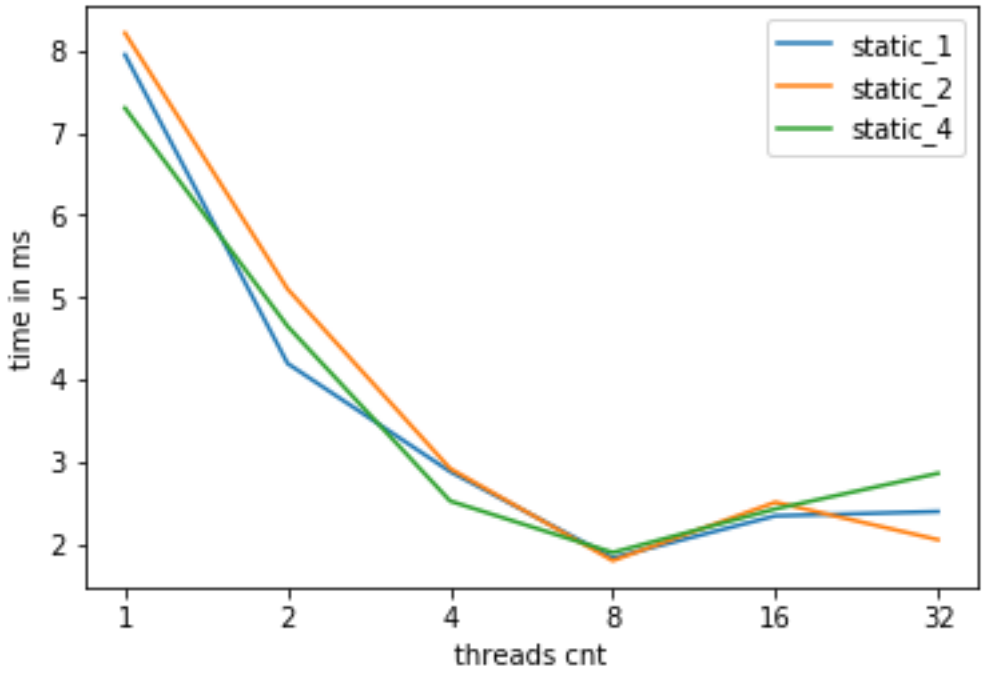

Рисунок 1 – Сравнение времени работы, в зависимости от chanks.

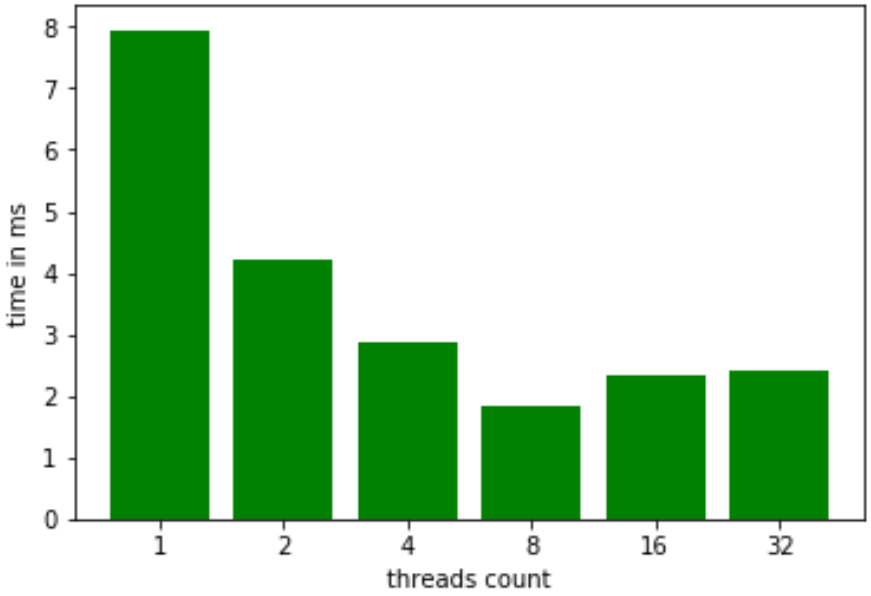

Рисунок 2 – Static, 1

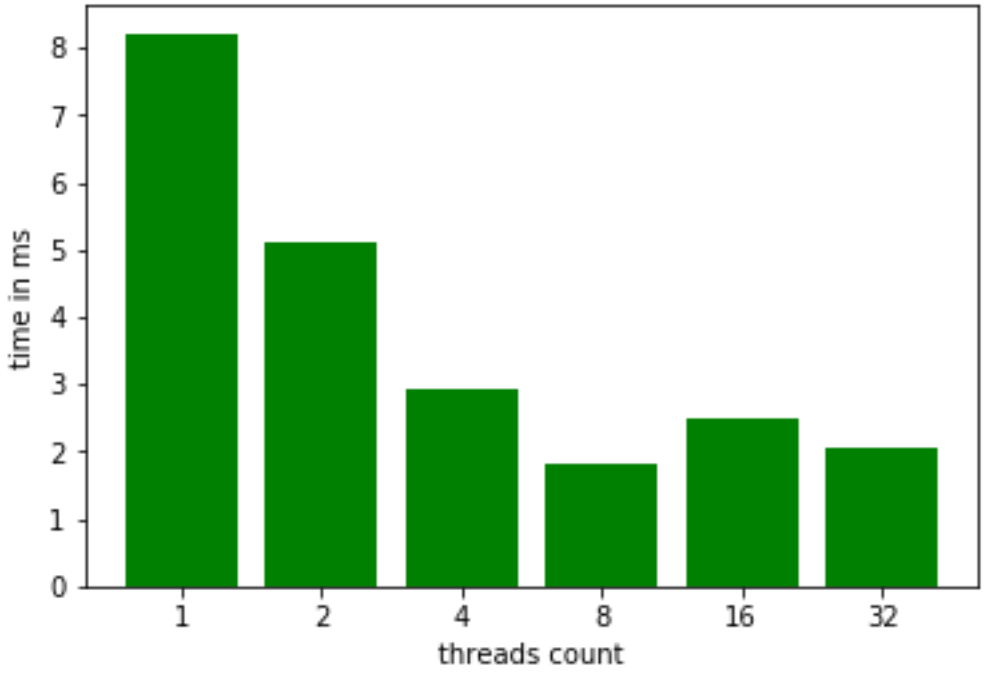

Рисунок 3 – Static, 2

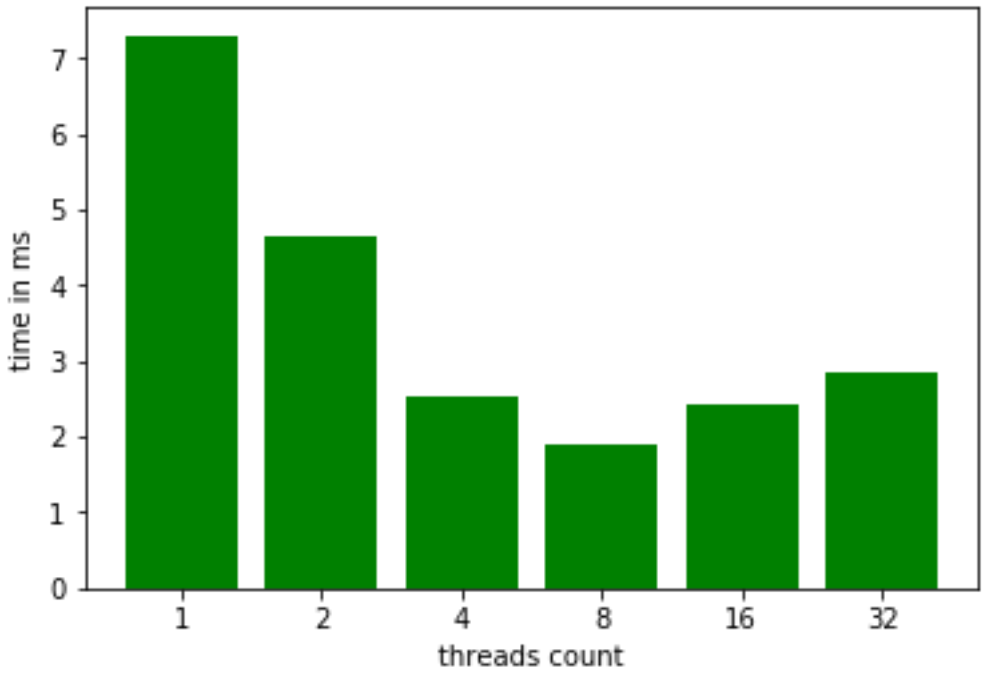

Рисунок 4 – Static, 4

Можно сделать вывод, что лучший результат времени работы программы получается при количестве тредов равному 8. Количество чанков влияет не особо.

Графики работы программы при schedule type = dynamic и chanks = {1, 2, 4}:

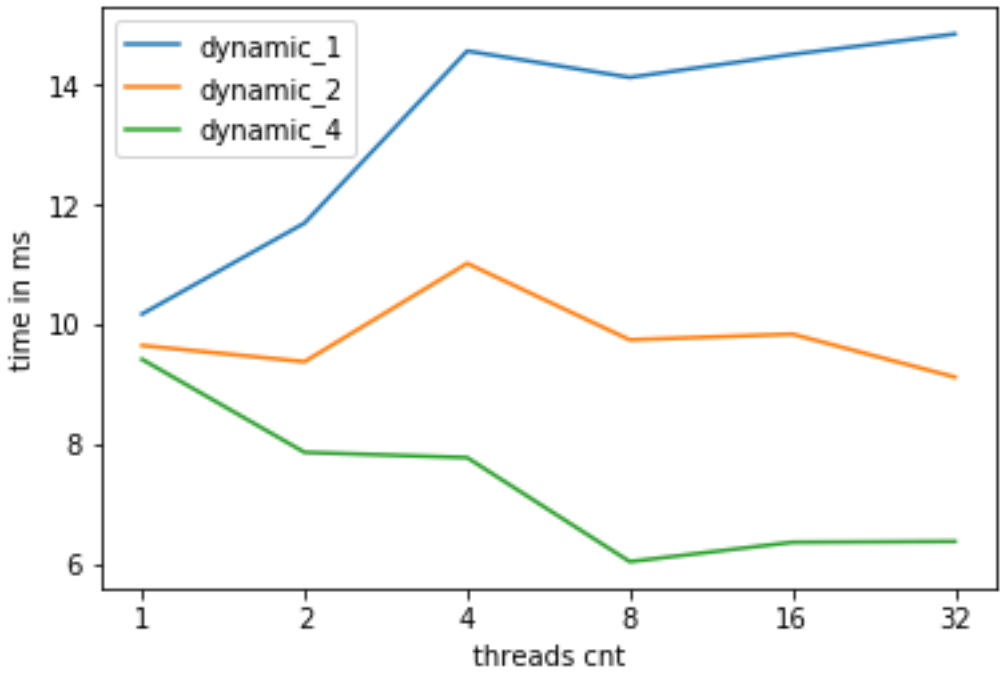

Рисунок 5 – Сравнение времени работы программы при dynamic, {1, 2, 4}

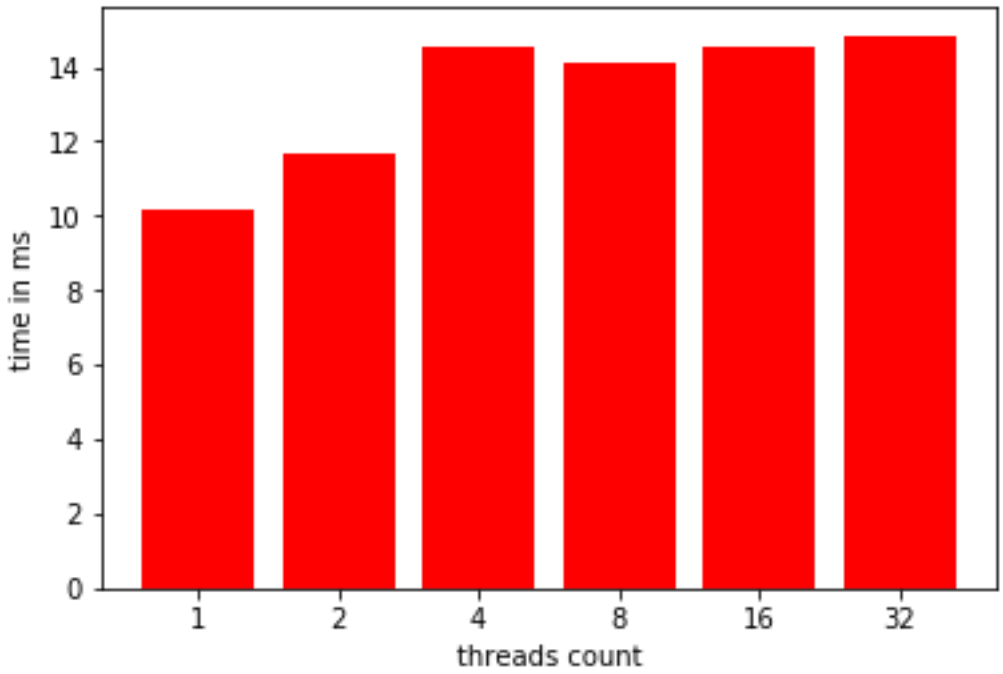

Рисунок 6 – Dynamic, 1

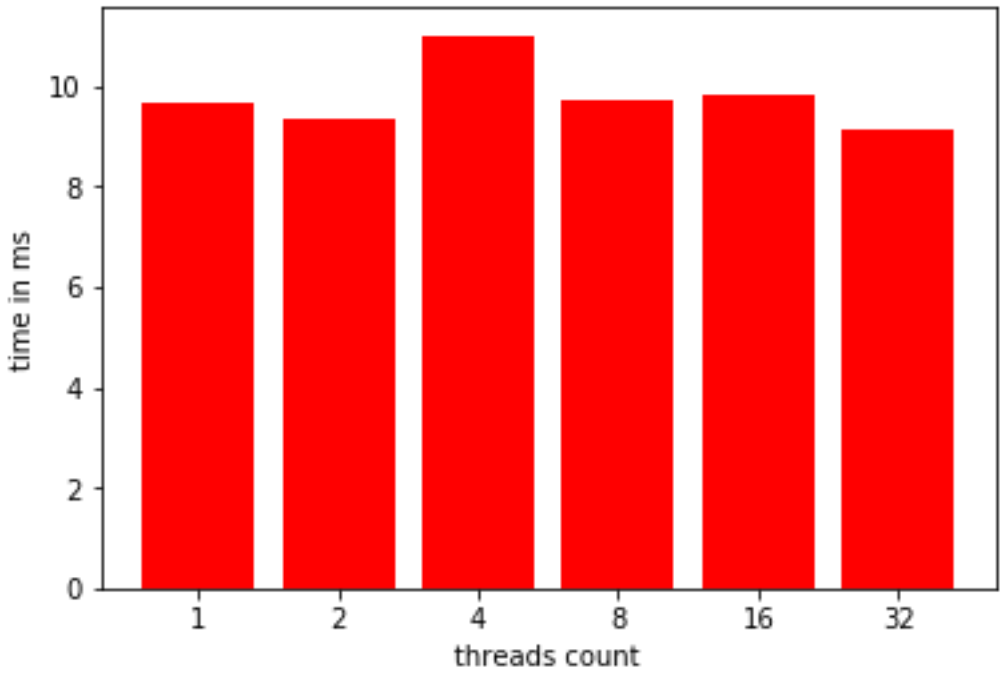

Рисунок 7 – Dynamic, 2

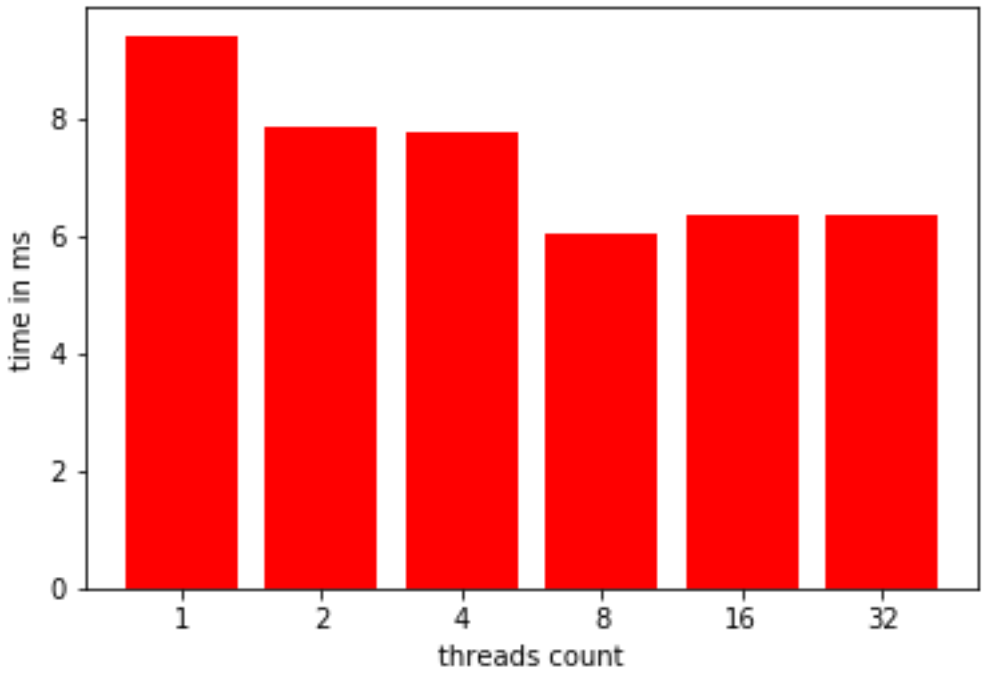

Рисунок 8 – Dynamic, 4

Можно сделать вывод, что конкретно в этой реализации программы лучше использовать schedule static, а не dynamic.

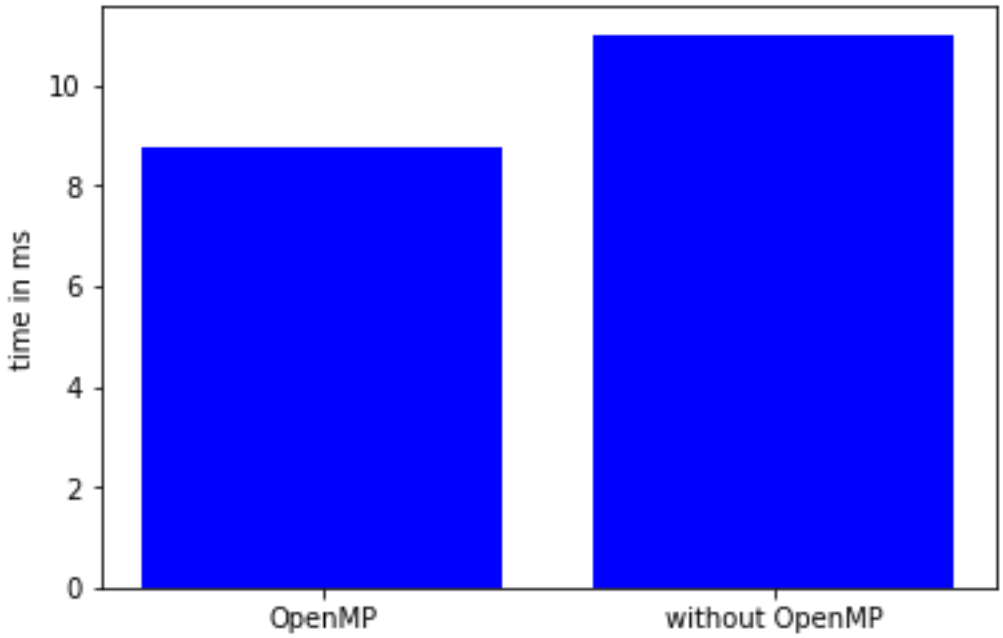

Рисунок 9 – Сравнение времени при одном треде и без OpenMP

Можно сделать вывод, что время работы программы без OpenMP и с ним при одном потоке отличается не очень сильно.

Листинг кода

hw5.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <algorithm>
#include <vector>
#include <fstream>
#include <omp.h>
using namespace std;

FILE* fp;
FILE* fo;
uint8_t c;

int new_val(double old, double minn, double maxx) {
    if (maxx > minn) {
```

```
        return (old / maxx - ((1 - old / maxx) / (1 - minn / maxx) * minn / maxx)) *
255;
    }
    return maxx;
}

void get_num(int& x) {
    while (true) {
        c = getc(fp);
        if (c == ' ' || c == '\n')
            return;
        x = x * 10 + (c - '0');
    }
}
void put_num(int x) {
    vector<uint8_t> a;
    while (x > 0) {
        a.push_back(x % 10 + '0');
        x /= 10;
    }
    for (int i = a.size() - 1; i >= 0; i--) {
        putc(a[i], fo);
    }
}
void put_header(char v, int w, int h) {
    putc('P', fo);
    putc(v, fo);
    putc('\n', fo);
    put_num(w);
    putc(' ', fo);
    put_num(h);
    putc('\n', fo);
    put_num(255);
    putc('\n', fo);
}
void get_header(string& ver, int& w, int& h, int& num, int& type) {
    c = getc(fp);
    ver += c;
    c = getc(fp);
    ver += c;
    c = getc(fp);
    type = (ver == "P6") ? 3 : 1;
    get_num(w);
    get_num(h);
    get_num(num);
}

int find_minim(vector<double>& val_cnt, double k) {
    for (int i = 0; i < 256; i++) {
        if (val_cnt[i] > k) {
            return i;
        }
        else {
            k -= val_cnt[i];
        }
```

```
    }
}

int find_maxim(vector<double>& val_cnt, double k) {
    for (int i = 255; i >= 0; i--) {
        if (val_cnt[i] > k) {
            return i;
        }
        else {
            k -= val_cnt[i];
        }
    }
}

string get_type_file(string name) {
    string type = "";
    bool ok = false;
    for (int i = 0; i < name.size(); i++) {
        if (name[i] == '.')
            ok = true;
        else if (ok)
            type += name[i];
    }

    return type;
}

bool is_support(string file) {
    return file == "ppm" || file == "pgm";
}

int main(int amount, char** args) {
    int threads_cnt = 0;
    for (int i = 0; i < 32; i++) {
        char n = args[1][i];
        if (!isdigit(n)) {
            break;
        }
        threads_cnt = threads_cnt * 10 + (n - '0');
    }


    double k = 0;
    cin >> k;

    omp_set_dynamic(0);
    if (threads_cnt != 0)
        omp_set_num_threads(threads_cnt);

    string file_one = get_type_file(args[2]);
    string file_two = get_type_file(args[3]);
    if (!is_support(file_one) || !is_support(file_two)) {
        cout << "this programm didn't support that type of files\n";
        return 0;
    }
```

```
    fp = fopen(args[2], "rb");
    if (fp == NULL) {
        cout << "Something goes wrong. Please check your file\n";
        return 0;
    }

    string ver;
    int w = 0, h = 0, num = 0, type = 0;
    get_header(ver, w, h, num, type);

    int image_size = w * h * type;
    vector<int> bytes(image_size);
    vector<double> val_cnt(256);
    int minim = 256;
    int maxim = -1;

    for (int i = 0; i < image_size; i++) {
        c = getc(fp);
        bytes[i] = c;
    }
    fclose(fp);

    double st = omp_get_wtime();

#pragma omp parallel
    {
        int cnt_private[256] = { 0 };

#pragma omp for schedule(static, 4)
        for (int i = 0; i < image_size; ++i) {
            cnt_private[bytes[i]]++;
        }
#pragma omp critical
        {
            for (int i = 0; i < 256; i++) {
                val_cnt[i] += cnt_private[i];
            }
        }
    }

    for (int i = 0; i < 256; i++) {
        val_cnt[i] /= image_size;
    }
    minim = find_minim(val_cnt, k);
    maxim = find_maxim(val_cnt, k);


    fo = fopen(args[3], "wb");
    put_header(ver == "P6" ? '6' : '5', w, h);

#pragma omp parallel for schedule(static, 4)
    for (int i = 0; i < image_size; i++) {
        bytes[i] = min(new_val(bytes[i], minim, maxim), 255);
```

```
    }


    double end = omp_get_wtime();
    printf("Time (%i thread(s): %g ms\n", threads_cnt, (end - st)*100);

    for (int i = 0; i < image_size; i++) {
        putc(bytes[i], fo);
    }
    fclose(fo);
}
```