САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №5

«OpenMP»

Выполнил(а): Левицкий Иван Михайлович

Номер ИСУ: 334916

студ. гр. М3135

Санкт-Петербург

2021

Цель работы: знакомство со стандартом OpenMP.

Инструментарий и требования к работе: рекомендуется использовать C, C++. Возможно использовать Python и Java. Стандарт OpenMP 2.0.

Теоретическая часть

OpenMP – открытый стандарт и набор инструментов для распараллеливания программ на C, C++ и Фортране. Даёт описание совокупности директив препроцессора (pragm), библиотечных процедур (например opm_set_num_threads) и переменных окружения, которые предназначены для программирования многопоточных программ. Одной из основных идей OpenMP является простота использования для программиста, т.к. ему не приходиться вручную возиться с синхронизацией потоков.

Команды OpenMP составляются в следующем порядке, сначала идут директивы, затем так называемые условия, при этом вся команда записывается как комментарий (через #), что позволяет запускать и компилировать как с подключенным OpenMP так и без него.

Для выполнения работы нам потребуются следующие директивы (и библиотечные процедуры) и условия:

1) omp_set_num_threads – библиотечная процедура, устанавливающая нужное число потоков.
2) parallel – указывает на то, что исполнение кода должно быть разделено на установленное число потоков.
3) for – указывает на то, что следующий цикл for должен быть распараллелен.

4) sections – указывает на то, что следующий блок {} будет разделен на секции (section) с помощью #pragma omp section, которые должны исполняться параллельно.
5) schedule – указывает на то, как должны распределяться потоки между ядрами. У нас для него есть всегда один или два параметра вида schedule(kind[chunk_size]) - kind in {‘static, ‘dynamic’}, chunk_size – целое положительное число. Static - изначально распределяет все потоки по ядрам, а dynamic - во время работы программы распределяет потоки на свободные ядра.
6) default – дает какой-то модификатор всем используемым переменным в следующем блоке (далее везде будет использоваться none, т.к. в последующих параметрах мы будем прописывать для надежности все условия вручную)
7) private – модификатор локальных переменных для всех потоков (в for i автоматически private).
8) shared – модификатор общих переменных для всех потоков.

Практическая часть

Алгоритм работы

1) Считываем входные данные из командной строки в формате <кол-во потоков> <имя входного файла> <имя выходного файла> <коэффициент игнорирования>. Затем последовательно обрабатываем ошибки некорректного ввода данных, проверяем корректность данных, файла и его заголовка (файл должен быть либо P6 (ppm), либо P5 (pgm); у него должны быть в заголовке поля width, height и maxBright, причем значение

maxBright должно равняться 255). Попутно мы также считали заголовок файла.

2) Далее разветвляемся в зависимости от того какой у нас файл ppm или pgm (и там и там дальнейшие действия аналогичны, так что опишем здесь только ppm). Считываем всю оставшуюся часть файла (то есть пиксели в формате подряд идущих rgb) и для удобства вывода распределяем их на 3 ветви (динамические массивы) по цвету (reds, greens, blues).

3) Выполняем по каждой ветви отдельно параллельно (для удобства и скорости раз мы их уже разделили) подсчет кол-ва значений каждого вида (от 0 до 255) в массивы sortRed, sortGreen, sortBlue. Далее выполняем поиск k-ой порядковой статистики сверху и снизу по каждому цвету на основе предыдущего подсчета и данного коэффициента во входных данных по (делаем это параллельно по 6 секциям получая значения maxWRed, maxWGreen, maxWBlue, maxBRed, maxBGreen, maxBBlue). Далее находим максимум из 3 полученных максимумов и минимум из 3 полученных минимумов, получая общие MAX и MIN по всем цветам.

4) Далее параллельно изменяем значение каждого вида цвета (от 0 до 255) по формуле:

$$newColour = \frac{(oldColour - MIN)}{MAX - MIN} \cdot 255$$

где newColour – новое значение цветового оттенка, oldColour – старое, MIN и MAX ранее найденные крайние значения игнорирования. То есть растягиваем все на диапазон [0, 255] с учетом игнорирования некоторого кол-ва значений сверху и снизу.

И затем, параллельно меняем уже каждое значение из считанных пикселей по только что рассчитанным видам значений от 0 до 255 и записываем их в

массивы для вывода по 3 цветовым каналам для удобства (outRed, outGreen, outBlue).

5) Записываем все в выходной файл в зависимости от того какой входной P5 или P6.

Замеры скорости работы

Все тестируется на корректной цветной картинке (P6 ppm) в 300 МБ. На тестирующей машине AMD Ryzen 7 с 8 ядрами и 16 виртуальными ядрами. Запуск провожу по 5 раз через VisualStudio на стандартном Microsoft VS компиляторе с ключом оптимизации по скорости O2 в Release и беру среднее значение.

Сначала, ради любопытства, проведем замер скорости на static при дефолтном значении chunk_size (см. рис 1), надеясь, что компилятор сам неплохо оптимизирует такой вариант (так и окажется впоследствии, хотя dynamic при не выставленном chunk_size я здесь даже не привожу, так как он работает очень медленно. Из этой зависимости легко заметить уже, что при увеличении кол-ва тредов скорость особенно по началу в среднем сильно возрастает, при 7-8 тредах достигает наибольшей отметки и дальше колеблется около нее с тенденцией к ухудшению, т.к. реальных ядер у нас 8.

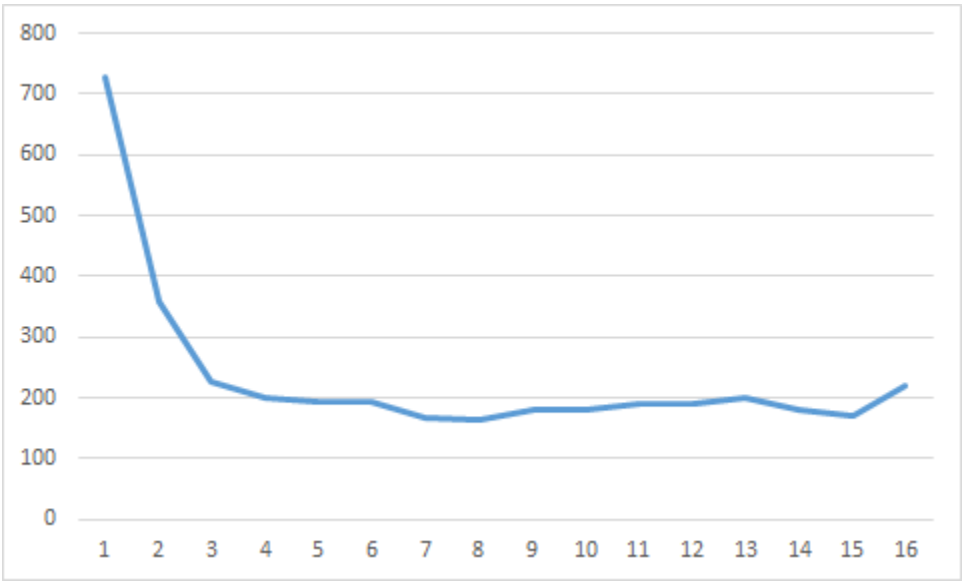

Рисунок №1 – График зависимости времени работы (шкала слева, в мс) от кол-ва потоков (шкала снизу) при параметре schedule static с неуказанным (default) chunk_size.

Далее замерим и static и dynamic при chunk_size 1 (см. рис 2).

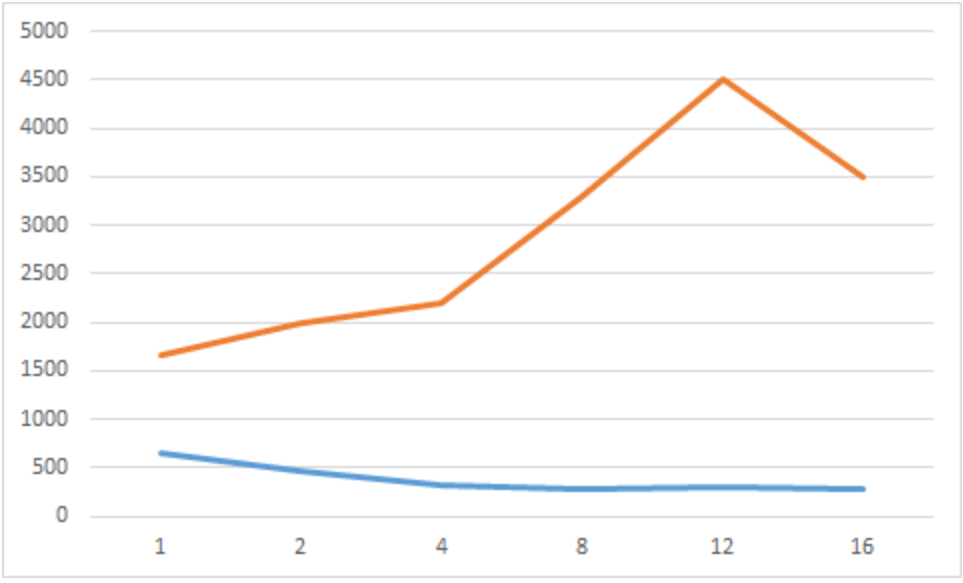

Рисунок №2 – График зависимости времени работы (шкала слева, в мс) от кол-ва потоков (шкала снизу) на static (синяя кривая) и dynamic (оранжевая кривая) при chunk_size 1 .

Из рисунка 2 видно, что static на порядки быстрее dynamic при одном значении и скорость при static возрастает, а при dynamic убывает.

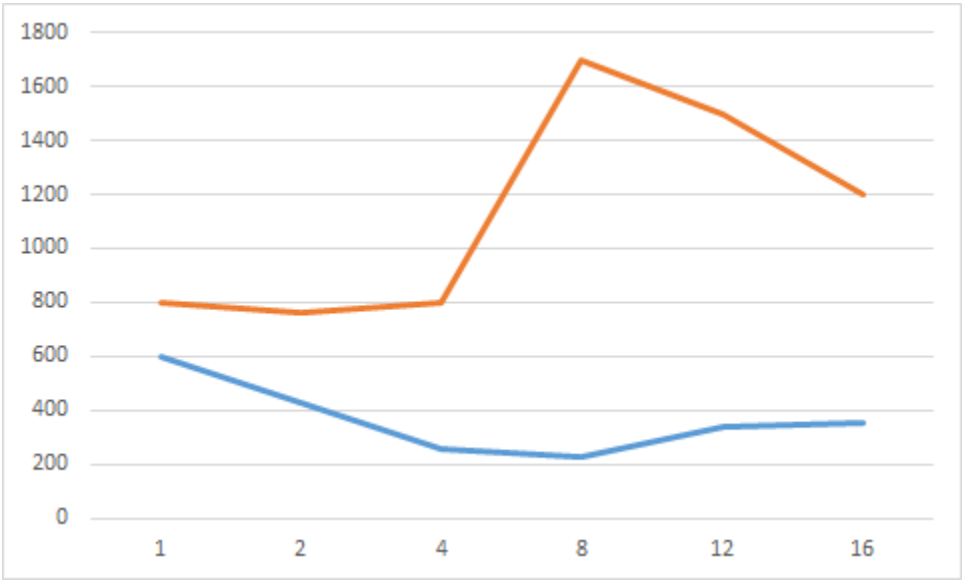

Рисунок №3 – График зависимости времени работы (шкала слева, в мс) от кол-ва потоков (шкала снизу) на static (синяя кривая) и dynamic (оранжевая кривая) при chunk_size 8.

Рисунок 3 согласуется с рисунком 2 по полученным зависимостям.

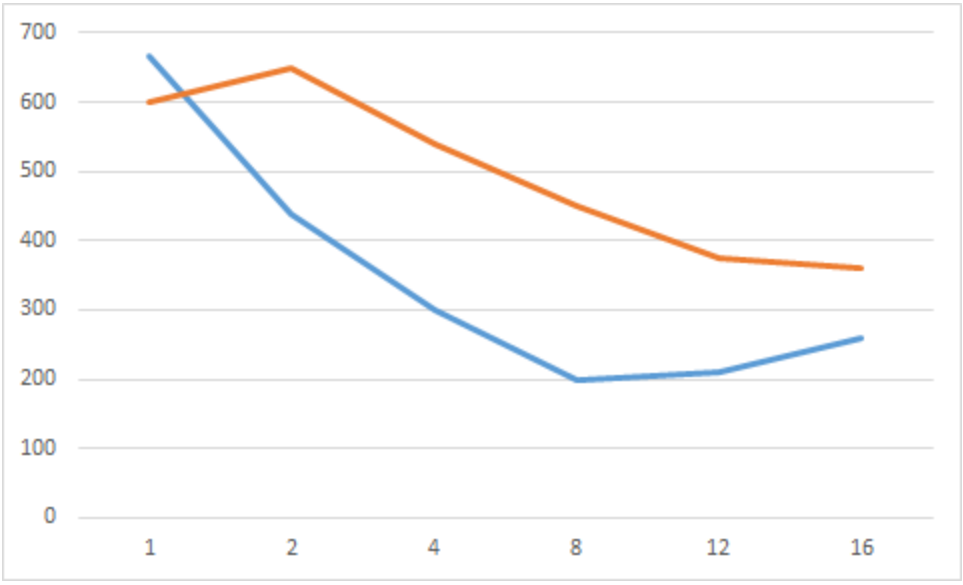

Рисунок №4 – График зависимости времени работы (шкала слева, в мс) от кол-ва потоков (шкала снизу) на static (синяя кривая) и dynamic (оранжевая кривая) при chunk_size 64.

Из рисунка 4 видно, что при chunk_size 64 dynamic начинает вести себя логичнее и уже ускоряется от увеличения числа потоков.

Далее проведем замеры на 8 потоках при разных значениях schedule (отдельно static и dynamic):

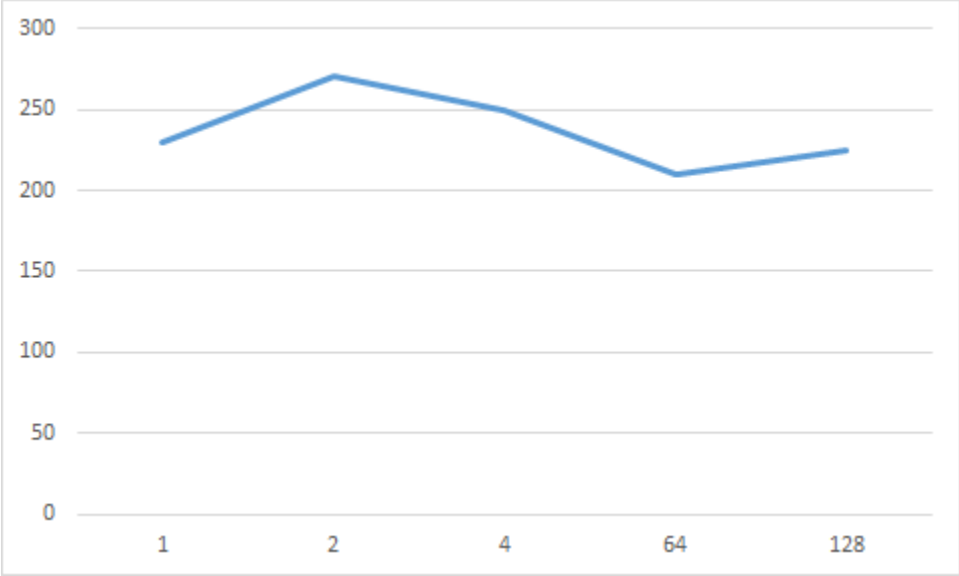

Рисунок №5 – График зависимости времени работы (шкала слева, в мс) от chunk_size (шкала снизу) на static.

Из рисунка 5 видно, что static на максимальном кол-ве логических ядер почти не зависит от параметра chunk_size.

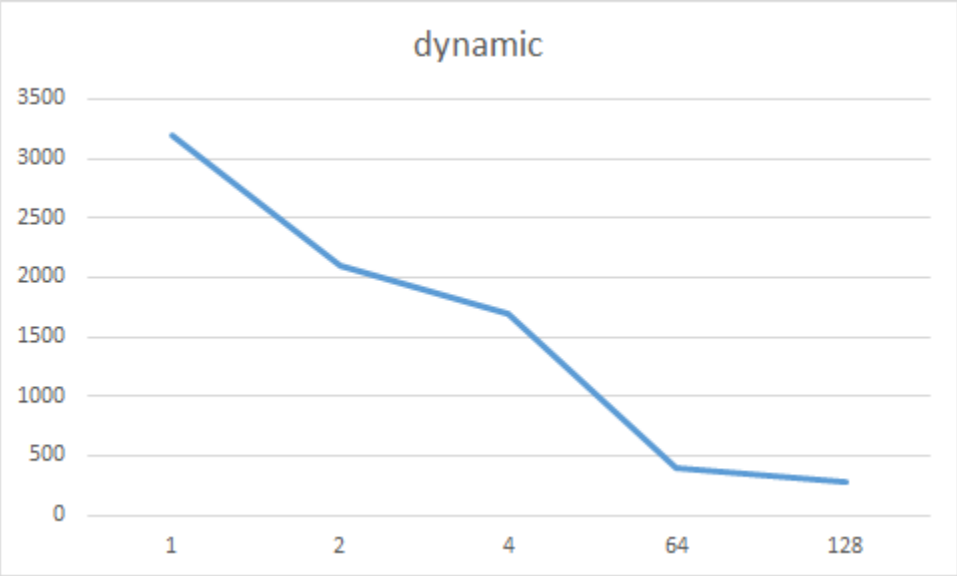

Рисунок №6 – График зависимости времени работы (шкала слева, в мс) от chunk_size (шкала снизу) на dynamic.

Из рисунка 6 видно, что dynamic очень сильно зависит от параметра chunk_size (что логично, т.к. чем больше у dynamic “пространства” тем лучше он распределяет очередь потоков)

Осталось провести замер зависимости с отключенным OpenMP и с включенным на 1 поток (сделаем это в несколько замеров и продемонстрируем их):

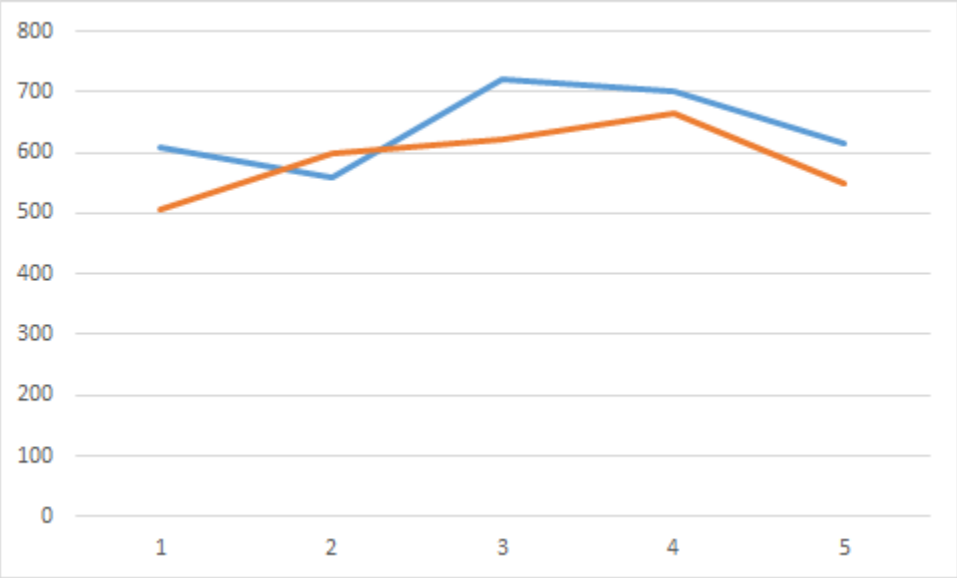

Рисунок №7 – График сравнения времени работы при наличии подключенного OpenMp (синяя кривая) и его отсутствии(оранжевая кривая), шкала снизу – номер запуска, слева – время в мс.

Из рисунка 7 очевидно, что OpenMP на 1 потоке слегка, но стабильно хуже выключенного OpenMP.

Листинг

ISO C++14, Компилятор: Microsoft Visual C++ 2019

```
#include <iostream>
#include <string>
#include <cmath>
#include <cstdio>
#include <fstream>
#include <cstdint>
#include <omp.h>
//#include <chrono>
#include <ctime>
using namespace std;

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

```
	double TimeStart;
	double TimeFinish;
	char* fileInputName;
	char* fileOutputName;
	float ratio; // переданный коэффициент игнорирования
	int countThreads = 0;
	int width = 0;
	int height = 0;
	int maxBright = -1;
	int partMaxBW; //кол-во значений которые нужно игнорировать
	int countPixels = 0; //кол-во пикселей width * height
	unsigned char *outReds = new unsigned char[1]; //массивы для записи уже измененных данных
	unsigned char *outGreens = new unsigned char[1];
	unsigned char *outBlues = new unsigned char[1];
	unsigned char *outGreys = new unsigned char[1];

	if (argc != 5) {
		cout << "You must enter your input in format: <countThreads> <fileInputName> <fileOutputName> <Ratio>" << endl;
		cout << "FileInput must be P6(PPM) or P5(PGM)" << endl;
		cout << "Ratio must be a float in range[0.0; 0.5)" << endl;
		cout << "countThreads must be an integer >= 0" << endl;
		return 1;
	}

	string str1 = argv[1];
	string str4 = argv[4];

	try { //кол-во потоков
		countThreads = stoi(argv[1]);
		if (countThreads < 0) {
			cout << "countThreads must be >= 0";
			return 1;
		}
```

```
	}
	catch (invalid_argument e) {
		cout << "countThreads isn't integer";
		return 1;
	}

	if (countThreads != 0) { //задаем кол-во потоков после их проверки
		omp_set_num_threads(countThreads);
	}

	try { //коэфициент
		ratio = stof(argv[4]);
		if (ratio >= 0.5 || ratio < 0) {
			cout << "Ratio must be in range[0.0, 0.5)";
			return 1;
		}
	}
	catch (invalid_argument e) {
		cout << "Ratio isn't float";
		return 1;
	}
	fileInputName = argv[2];
	fileOutputName = argv[3];
	ifstream inImage;
	inImage.open(fileInputName, ios::binary);
	if (!inImage.is_open()) {
		cout << "File: " + string(fileInputName) + " cannot read and/or not
found";
		return 1;
	}

	string typeFile;
	try { // общее считывание хедера и обработка ошибок
		getline(inImage, typeFile);
```

```
	if (typeFile != "P5" && typeFile != "P6") { //магическое число
		cout << "FileInput isn't P6 and isn't P5 format PNM" << endl;
		inImage.close();
		return 1;
	}
	string strBuf = "";
	char ch;

	while (!inImage.eof()) { // width
		inImage.get(ch);
		if (!isdigit(ch)) {
			break;
		}
		else {
			strBuf += ch;
		}
	}
	width = stoi(strBuf);

	strBuf = "";
	while (!inImage.eof()) { // height
		inImage.get(ch);
		if (!isdigit(ch)) {
			break;
		}
		else {
			strBuf += ch;
		}
	}
	height = stoi(strBuf);

	if (height <= 0 || width <= 0) {
		cout << "FileInput uncorrectly, width and height image must be
> 0";
```

```
                return 1;
            }
            countPixels = width * height; // общее кол-во пикселей
            partMaxBW = int(ratio * width * height); // задаем кол-во пикселей которое нужно игнорировать

            strBuf = "";
            while (!inImage.eof()) { // maxBright
                inImage.get(ch);
                if (!isdigit(ch)) {
                    break;
                }
                else {
                    strBuf += ch;
                }
            }
            maxBright = stoi(strBuf);

            if (maxBright != 255) {
                cout << "FileInput uncorrectly, maximum bright must be 255";
                return 1;
            }

            if (typeFile == "P5") { // разветвление на pgm и ppm
//pgm
                unsigned char *greys = new unsigned char[countPixels]; //все считываемые пиксели

                for (int i = 0; i < countPixels; i++) {
                    inImage.get(ch);
                    greys[i] = ch;
                }
                inImage.close();
                TimeStart = clock();
```

```
                    int* sortGrey = new int[256]; // массив для сортировки по 256
значениям пикселя
                    for (int i = 0; i < 256; i++) {
                           sortGrey[i] = 0;
                    }

                    for (int i = 0; i < countPixels; i++) {// подсчет
                           sortGrey[greys[i]]++;
                    }

                    int maxBGrey = 0, maxWGrey = 0; //минимум и максимум
соответственно

                    int minPix = 0;
                    int maxPix = 255;
                    int sum = 0;
                    #pragma omp parallel sections private(sum) shared(partMaxBW,
sortGrey, minPix, maxPix, maxBGrey, maxWGrey)
                    {      //поиск k-ой порядковой стастистики сверху и снизу на
основе подсчета выше
                           #pragma omp section
                           {
                                  sum = 0;
                                  while (sum < partMaxBW && minPix < 256) {
                                         sum += sortGrey[minPix];
                                         minPix++;
                                  }
                                  maxBGrey = max(0, minPix - 1);
                           }
                           #pragma omp section
                           {
                                  sum = 0;
                                  while (sum < partMaxBW && maxPix >= 0) {
                                         sum += sortGrey[maxPix];
                                         maxPix--;
                                  }
```

```
                        maxWGrey = min(255, maxPix + 1);
                    }
                }
                if (maxWGrey == maxBGrey) {
                    #pragma omp parallel for schedule(static) default(none)
shared(greys, countPixels, maxBGrey, outGreys)
                    for (int i = 0; i < countPixels; i++) {
                        if (greys[i] <= maxBGrey) {
                            outGreys[i] = 0;
                        }
                        else {
                            outGreys[i] = 255;
                        }
                    }
                }
                else {
                    float gr = 255.0 / (maxWGrey - maxBGrey);
                    float grd = gr * maxBGrey;
                    outGreys = new unsigned char[countPixels];
                    #pragma omp parallel for schedule(static) default(none)
shared(gr, grd, sortGrey, greys)
                    for (int i = 0; i < 256; i++) { //изменение каждого
вида пикселя (по его числовому значению)
                        sortGrey[i] = min(255, max(0, int(round(i * gr -
grd))));
                    }
                    #pragma omp parallel for schedule(static) default(none)
shared(gr, grd, greys, countPixels, sortGrey, outGreys)
                    for (int i = 0; i < countPixels; i++) { //изменение
кадого пикселя
                        outGreys[i] = sortGrey[greys[i]];
                    }
                }
                TimeFinish = clock();
                delete[] sortGrey;
                delete[] greys;
```

```
		}
		else {
//ppm
			unsigned char *reds = new unsigned char[countPixels];
//массивы всех считанных пикселей по 3 цветовым каналам
			unsigned char *greens = new unsigned char[countPixels];
			unsigned char *blues = new unsigned char[countPixels];;
			for (int i = 0; i < countPixels * 3; i++) {
				inImage.get(ch);
				if (i % 3 == 0) {
					reds[i / 3] = ch;
				}
				else if (i % 3 == 1) {
					greens[i / 3] = ch;
				}
				else {
					blues[i / 3] = ch;
				}
			}
			inImage.close();
//ищем крайние значения которые нужно игнорировать
			TimeStart = clock();
			int *sortRed = new int[256]; //массивы для сортировки по 256
значениям по каждому цвету
			int *sortGreen = new int[256];
			int *sortBlue = new int[256];
			for (int i = 0; i < 256; i++) {
				sortRed[i] = 0;
				sortGreen[i] = 0;
				sortBlue[i] = 0;
			}
#pragma omp parallel sections
			{
				#pragma omp section
				{
```

```
                        for (int i = 0; i < countPixels; i++) { //подсчет
                                sortRed[reds[i]]++;
                        }
                }
                #pragma omp section
                {
                        for (int i = 0; i < countPixels; i++) { //подсчет
                                sortGreen[greens[i]]++;
                        }
                }
                #pragma omp section
                {
                        for (int i = 0; i < countPixels; i++) { //подсчет
                                sortBlue[blues[i]]++;
                        }
                }
        }
        int maxBRed = 0, maxWRed = 0, maxBGreen = 0, maxWGreen = 0,
maxBBlue = 0, maxWBlue = 0; //соответственно макс и мин по каждому цвету (потом
найдем общие)


        int minPix = 0;
        int maxPix = 255;
        int sum = 0;
    #pragma omp parallel sections default(none) private(sum, minPix, maxPix)
shared(maxWRed, maxWGreen, maxWBlue, maxBRed, maxBGreen, maxBBlue, sortRed,
sortGreen, sortBlue)
        {// поиск k-ой порядковой статистики по каждому цвету сверху и
снизу (W(white) сверху, B(black) снизу
                #pragma omp section
                {
                        minPix = 0;
                        sum = 0;
                        while (sum < partMaxBW && minPix < 256) {
                                sum += sortRed[minPix];
                                minPix++;
```

```
		}
		maxBRed = max(0, minPix - 1);
	}
	#pragma omp section
	{
		minPix = 0;
		sum = 0;
		while (sum < partMaxBW && minPix < 256) {
			sum += sortGreen[minPix];
			minPix++;
		}
		maxBGreen = max(0, minPix - 1);
	}
	#pragma omp section
	{
		minPix = 0;
		sum = 0;
		while (sum < partMaxBW && minPix < 256) {
			sum += sortBlue[minPix];
			minPix++;
		}
		maxBBlue = max(0, minPix - 1);
	}
	#pragma omp section
	{
		maxPix = 255;
		sum = 0;
		while (sum < partMaxBW && maxPix >= 0) {
			sum += sortRed[maxPix];
			maxPix--;
		}
		maxWRed = min(255, maxPix + 1);
	}
	#pragma omp section
```

```
                        {
                                maxPix = 255;
                                sum = 0;
                                while (sum < partMaxBW && maxPix >= 0) {
                                        sum += sortGreen[maxPix];
                                        maxPix--;
                                }
                                maxWGreen = min(255, maxPix + 1);
                        }
                        #pragma omp section
                        {
                                maxPix = 255;
                                sum = 0;
                                while (sum < partMaxBW && maxPix >= 0) {
                                        sum += sortBlue[maxPix];
                                        maxPix--;
                                }
                                maxWBlue = min(255, maxPix + 1);
                        }
                }
                outReds = new unsigned char[countPixels];
                outGreens = new unsigned char[countPixels];
                outBlues = new unsigned char[countPixels];
                int MAX = max(max(maxWRed, maxWGreen), maxWBlue); //находим
общий максимум и минимум
                int MIN = min(min(maxBRed, maxBGreen), maxBBlue);
                if (MAX == MIN) {
#pragma omp parallel for schedule(static) default(none) shared(countPixels,
outReds, outGreens, outBlues, reds, greens, blues, MIN)
                        for (int i = 0; i < countPixels; i++) {
                                if (reds[i] <= MIN) {
                                        outReds[i] = 0;
                                }
                                else {
                                        outReds[i] = 255;
```

```
                                }
                                if (greens[i] <= MIN) {
                                        outGreens[i] = 0;
                                }
                                else {
                                        outGreens[i] = 255;
                                }
                                if (blues[i] <= MIN) {
                                        outBlues[i] = 0;
                                }
                                else {
                                        outBlues[i] = 255;
                                }
                        }
                }
                else {
                        float m = 255.0 / (MAX - MIN); // для удобства выделяем составные части формулы изменения
                        float d = m * MIN;
#pragma omp parallel for schedule(static) default(none) shared(m, d, sortRed, sortGreen, sortBlue)
                        for (int i = 0; i < 256; i++) { //изменение каждого вида пикселя (по его числовому значению)
                                sortRed[i] = min(255, max(0, int(round(i * m - d)))); //для удобства записываем в те же массивы
                                sortGreen[i] = min(255, max(0, int(round(i * m - d))));
                                sortBlue[i] = min(255, max(0, int(round(i * m - d))));
                        }
#pragma omp parallel for schedule(static) default(none) shared(countPixels, outReds, outGreens, outBlues, sortRed, sortGreen, sortBlue, reds, greens, blues)
                        for (int i = 0; i < countPixels; i++) { //изменение кадого пикселя
                                outReds[i] = sortRed[reds[i]];
                                outGreens[i] = sortGreen[greens[i]];
                                outBlues[i] = sortBlue[blues[i]];
```

```
                        }
                }
                TimeFinish = clock();
                delete[] reds;
                delete[] greens;
                delete[] blues;
                delete[] sortRed;
                delete[] sortGreen;
                delete[] sortBlue;
        }
    }
    catch (exception e) { //проверка на всевозможные ошибки в ходе чтения из-за некоректности файла
        cout << "FileInput uncorrectly" << endl;
        cout << "FileInput must be coorectly P6(PPM) or P5(PGM) without comments";
        return -2;
    }
//вывод
    ofstream outImage;
    outImage.open(fileOutputName, ios::binary);
    if (!outImage.is_open()) {
        cout << "This output file " << fileOutputName << "is incorrect or not found";
        return 1;
    }
    outImage << typeFile << endl;
    outImage << width << " " << height << endl;
    outImage << 255 << endl;
    if (typeFile == "P5") {
        for (int i = 0; i < countPixels; i++) {
            outImage << outGreys[i];
        }
    }
    else {
```

```
			for (int i = 0; i < countPixels; i++) {
				outImage << outReds[i] << outGreens[i] << outBlues[i];
			}
		}
		outImage.close();
		printf("Time (%i thread(s)) : %g ms\n", countThreads, TimeFinish - TimeStart); //время (по ctime) в мс
		delete[] outReds;
		delete[] outBlues;
		delete[] outGreens;
		delete[] outGreys;
		return 0;
}
```