САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №5

«OpenMP»

Выполнил(а): Белозоров Денис Сергеевич

Номер ИСУ: 334876

студ. гр. М3139

Санкт-Петербург

2022

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 (Open Multi-processing) - открытый стандарт для распараллеливания программ на языках C, C++ и Fortran.

Задачи, выполняемые с помощью OpenMP, описываются прагмами(#pragma omp), процедурами(omp_) и переменными окружения. Такая архитектура предоставляет достаточную простоту реализации и легко позволяет выполнить программу без OpenMP.

При компиляции необходимо указать флаг -fopenmp.

OpenMP реализует параллельные вычисления с помощью многопоточности (модель fork-join) - главный поток порождает несколько дополнительных, между которыми разделяется задача. (см. Рисунок 1)

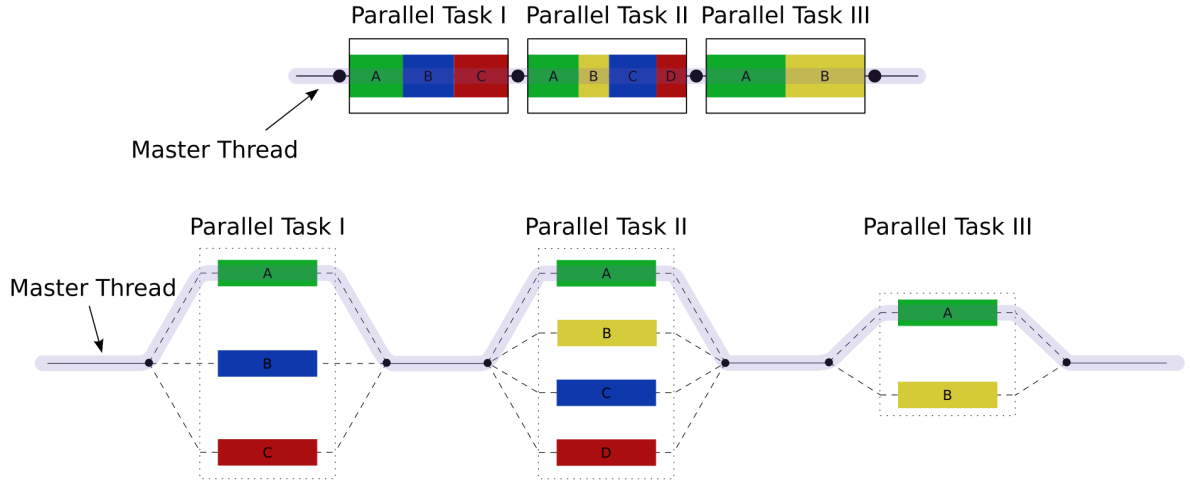

Рисунок 1 - иллюстрация многопоточности

Использовались следующие компоненты OpenMP:

for - указание оптимизировать цикл, распределяя его между потоками

parallel - указание выполнять несколькими потоками

shared - обозначение переменных, общих для области, выполняемой параллельно

critical - обозначение области, исполняемой только одним потоком в одним момент времени

schedule - обозначение метода разбиения цикла на потоки.

- static, chunk_size разбивает цикл на блоки по chunk_size. Если не указан chunk_size, то разбиение на приблизительно равные блоки. Блоки статично назначаются потокам.
- dynamic, chunk_size разбивает цикл на блоки по chunk_size. Если не указан chunk_size, то он считается равным 1. Каждому потоку назначается выполнение своего блока. Когда поток завершает выполнение, ему динамически назначается новый блок.

omp_set_num_threads - процедура, устанавливающая число потоков, используемых в параллельных областях.

omp_get_max_threads - процедура, возвращающая максимальное число потоков, доступное для использования в параллельной области

omp_get_wtime - процедура, возвращающая число секунд, прошедших с некоторого момента времени.

Практическая часть

Задание - hard.

Для коррекции контрастности был выбран следующий алгоритм:

1) Прочитаем изображение, P5 или P6.
2) Применим сортировку подсчетом по каждому из каналов r, g, b.

3) Рассчитаем del = width * height * cof - число значений, которое нужно проигнорировать.
4) Для каждого канала из результатов сортировки подсчетом вычтем del максимальных и минимальных значений
5) Таким образом, останется только взять минимальное и максимальное значение, но уже по всем каналам.
6) Зная соответственно mn и mx можно нормализовать значения в каждой точке:

   rgb[i][g] = min(255, max(0, 255 * (rgb[i][g] - mn) / (mx - mn)))

7) Запишем полученное изображение.

Примеры работы программы:

Входной файл - P6, 6960 на 4640 пикселей (см. рисунок 2):

Рисунок 2 - первый входной файл

Выходной файл, коэффициент = 0.1 (см. рисунок 3):

Рисунок 3 - первый выходной файл

Входной файл - P5, 512 на 512 пикселей (см. рисунок 4):

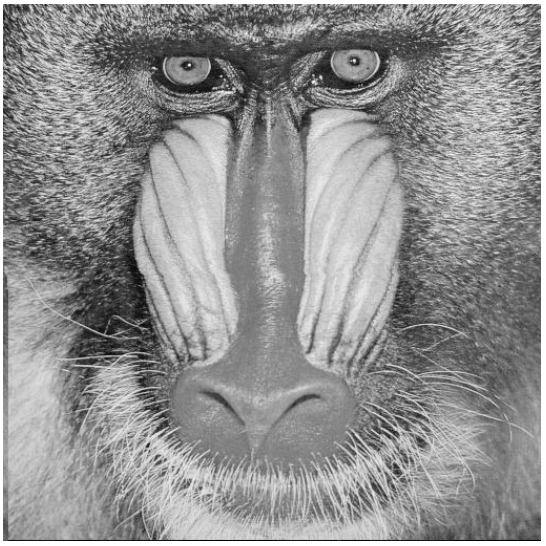

Рисунок 4 - второй входной файл

Выходной файл, коэффициент = 0.1 (см. рисунок 5):

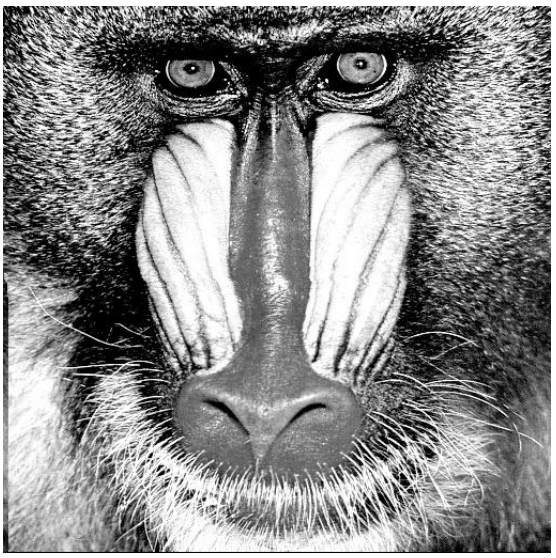

Рисунок 5 - второй выходной файл

Графики:

Все данные были получены при обработке изображения типа P6, размером 6960 на 4640 пикселей. (см рисунок 2 - первый входной файл).

График зависимости времени работы в секундах от типа schedule. (см Рисунок 6)

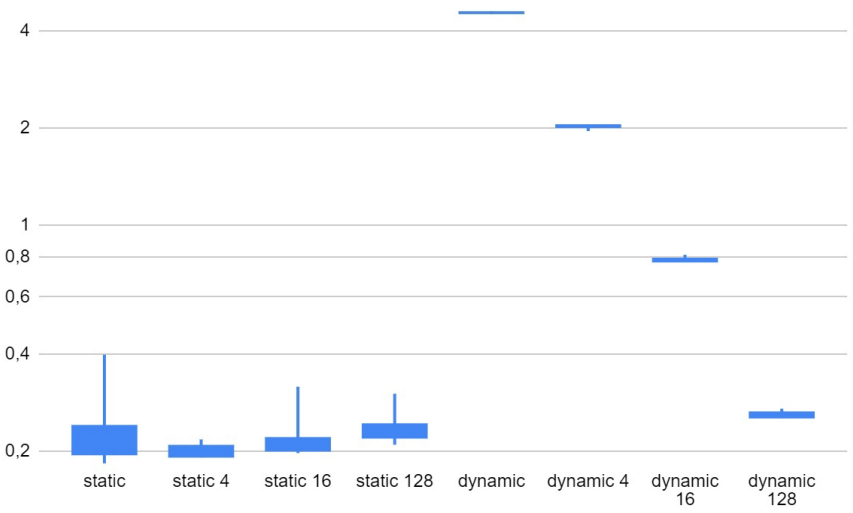

Рисунок 6 - график при различных schedule

График зависимости времени в секундах от числа потоков при schedule(static, 4). (см. Рисунок 7).

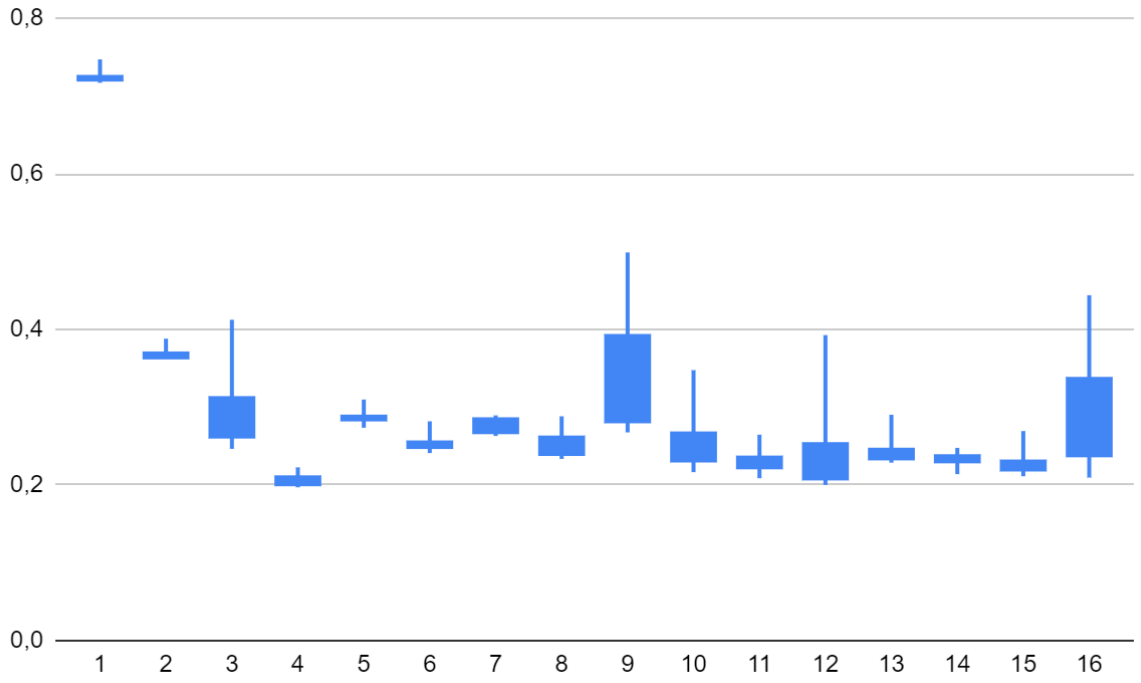

Рисунок 7 - график при различном числе потоков

График зависимости времени в секундах от включенности openmp. (см. Рисунок 8).

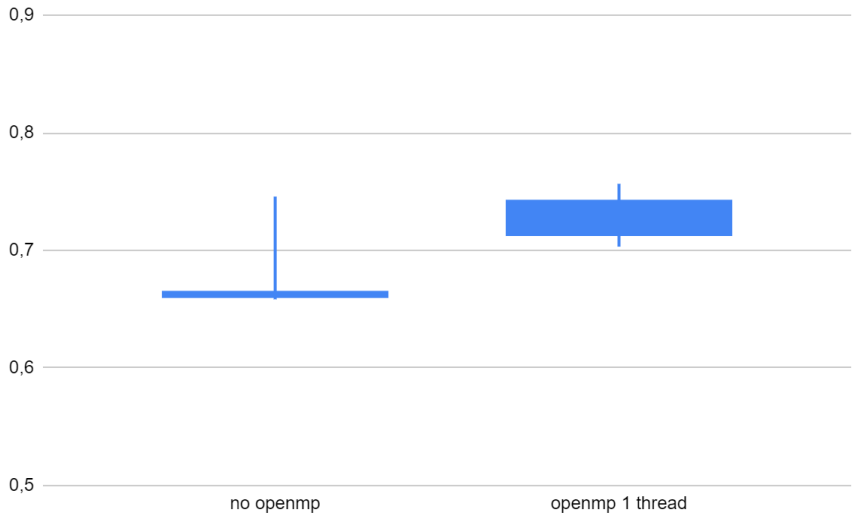

Рисунок 8 - график при включенном\выключенном Openmp

Листинг

Язык программирования - C++, Компилятор - g++ 9.4.0

solve.cpp

```
#include <stdio.h>
#include <omp.h>
#include <fstream>
#include <chrono>
#define min(a, b) a <= b ? a : b
#define max(a, b) a >= b ? a : b
#define MODE static, 4

struct Image{
    unsigned char* data;
    int width;
    int height;
    bool isp5;
```

```
    ulong count;

    Image(int width, int height, bool isp5){
        this->width = width;
        this->height = height;
        this->isp5 = isp5;
        this->count = (ulong)(isp5 ? 1 : 3) * height * width;
        this->data = (unsigned char*)malloc(count * sizeof(unsigned
char));
    }
    Image(){}
};


Image readImage(FILE *fp){
    getc(fp);
    bool p5 = false;
    if(getc(fp) == 53){
        p5 = true;
    }

    getc(fp);
    int width, height, grayscale;
    fscanf(fp, "%d%d%d", &width, &height, &grayscale);
    getc(fp);

    Image result(width, height, p5);
    fread(result.data, sizeof(unsigned char), result.count, fp);
    return result;
}

void printImage(Image img, FILE *fp){
    if(img.isp5){
        fputs((std::string("P5\n") + std::to_string(img.width) + " " +
std::to_string(img.height) + "\n" + std::to_string(255) + "\n").c_str(),
fp);
    }else{
        fputs((std::string("P6\n") + std::to_string(img.width) + " " +
std::to_string(img.height) + "\n" + std::to_string(255) + "\n").c_str(),
fp);
    }
    fwrite(img.data, sizeof(unsigned char), img.count, fp);
    free(img.data);
}

void handleImage(Image img, double coff){
    ulong deleteCount = (ulong)(img.width) * img.height * coff;
```

```
ulong counts[256][3] = {0};
// sort
#pragma omp parallel shared(counts, img)
{
    ulong buffer[256][3] = {0};
    #pragma omp for schedule(MODE)
    for(int i = 0 ;i < img.count;i++){
        if(img.isp5){
            buffer[img.data[i]][0]++;
        }else{
            buffer[img.data[i]][i % 3]++;
        }
    }
    #pragma omp critical
    for(int i = 0; i < 256;i++){
        for(int g = 0; g < 3;g++){
            counts[i][g] += buffer[i][g];
        }
    }
}
// min/max
int mn = 255;
int mx = 0;

for(int g = 0; g < 3 || g < 1 && img.isp5;g++){
    int ost = deleteCount;
    for(int i = 0; i < 256;i++){
        int v = min(ost, counts[i][g]);
        ost -= counts[i][g];
        counts[i][g] -= v;
        if(ost < 0){
            mn = min(mn, i);
            break;
        }
    }
    ost = deleteCount;
    for(int i = 255; i >= 0;i--){
        int v = min(ost, counts[i][g]);
        ost -= counts[i][g];
        counts[i][g] -= v;
        if(ost < 0){
            mx = max(mx, i);
            break;
        }
    }
}
if(mn >= mx){
```

```
        printf("Image consists only of one color or coefficient is too
big. Image was not handled.\n");
        return;
    }
    // normalize
    #pragma omp parallel for schedule(MODE) shared(img, mn, mx)
    for(int i = 0 ;i < img.count;i++){
        int value = img.data[i];
        img.data[i] = (value < mn) ? (0) : ((value > mx) ? 255 : (255 *
(value - mn)) / (mx - mn));
    }

}

int main(int argc, char **argv){
    if(argc != 5){
        printf("Wrong arguments count, expected: thread count, input,
output, coefficient\n");
        return 0;
    }

    int thread_count;
    try{
        thread_count = std::atoi(argv[1]);
        if(thread_count < 0){
            printf("Thread count can not be negative\n");
            return 0;
        }
    }catch(...){
        printf("Wrong argument - thread count\n");
        return 0;
    }

    #ifdef _OPENMP
        if(thread_count != 0){
            omp_set_num_threads(thread_count);
        }else{
            thread_count = omp_get_max_threads();
            printf("Default number of threads is %d\n", thread_count);
        }
    #endif
    Image image;
    try{
        FILE *fp = fopen(argv[2], "rb");
        if(!fp){
            printf("Input file is not accessible\n");
            return 0;
```

```
        }
        image = readImage(fp);
        fclose(fp);
    }catch(...){
        printf("input file is not correct\n");
    }
    double coff;
    try{
        coff = std::atof(argv[4]);
        if(coff < 0 || coff >= 0.5){
            printf("Incorrect coefficient value\n");
            return 0;
        }
    }catch(...){
        printf("Wrong argument - coefficient\n");
        return 0;
    }

    double start;
    auto startChrono = std::chrono::steady_clock::now();
    #ifdef _OPENMP
        start = omp_get_wtime();
    #endif
    handleImage(image, coff);
    #ifdef _OPENMP
        printf("Time (%i thread(s)): %g ms\n", thread_count,
omp_get_wtime() - start);
    #else
        printf("Time: %g ms\n",
std::chrono::duration<double>(std::chrono::steady_clock::now() -
startChrono).count());
    #endif

    try{
        FILE* fp;
        fp = fopen(argv[3], "wb");
        if(!fp){
            printf("Output file is not accessible\n");
            return 0;
        }
        printImage(image, fp);
        fclose(fp);
    }catch(...){
        printf("Can not save image\n");
    }
}
```