САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИТМО
Дисциплина: Архитектура ЭВМ

Отчет
по домашней работе №5
«OpenMP»

Выполнил(а): Альжанов Максим Булатович
студ. гр. М3139

Санкт-Петербург
2020

Цель работы: знакомство со стандартом распараллеливания команд OpenMP.

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

Теоретическая часть

OpenMP – стандарт для распараллеливания программ. Даёт описание для директив компилятора (pragm-ы), библиотечных процедур (начинаются на omp_) и переменных окружения, которые предназначены для создания программ, разделяемых на несколько потоков. OpenMP вводит параллелизм в ваше приложения запуская несколько потоков которые исполняют код параллельно. Для распараллеливания используются pthreads (стандарт используемый в POSIX системах), которые можно было бы запускать вручную. Преимущество использования OpenMP в том, что программисту не нужно писать код, синхронизирующий несколько потоков вручную.

Все директивы OpenMP вставляются в код как комментарии что позволяет безболезненно компилировать программы компиляторами которые не поддерживают стандарт OpenMP или если это не требуется (нет флага `-fopenmp`) и тогда код выполняться без распараллеливания – последовательно.

Когда исполнение входит в параллельный регион OpenMP – мастер-поток создаёт группу потоков (с помощью fork-а) (см Рисунок 1 - Работа OpenMP). Когда исполнение доходит до конца такого региона - происходит синхронизация. Это называется fork-join параллелизм. При этом поток-предок не завершается, а остаётся открытым для последующего использования с целью экономии времени.

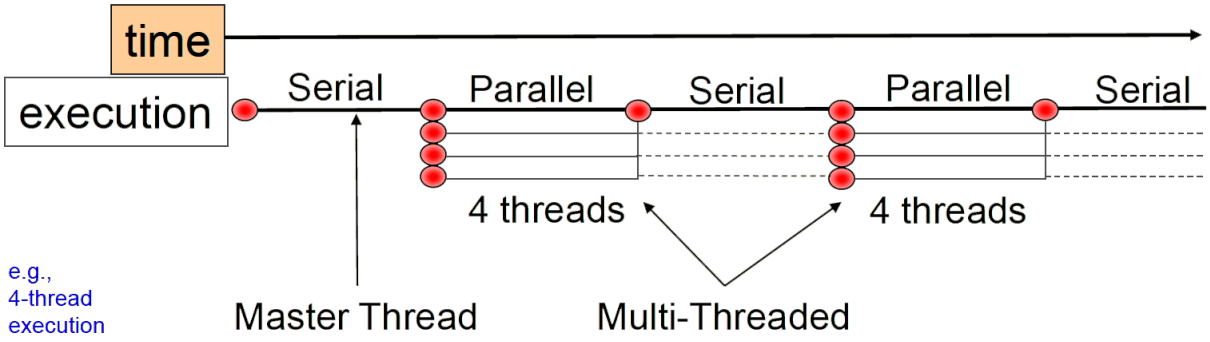

Рисунок 1 - Работа OpenMP

Директива OpenMP состоит из «стража» (sentinel), который интерпретируется компилятором либо как комментарий, либо как начало директивы OpenMP, за которым следует сама директива и условие (clause). Для языков c и c++ стражем является `#pragma omp`. Чтобы компилятор начал воспринимать эти директивы необходимо добавить флаг `-fopenmp`.

В своей работе я использовал следующие директивы:

`parallel` – указывает на то что код должен выполняться на всех ядрах параллельно.

`for` – указывает, что код который исполняется циклом for должен быть распределён между потоками

Помимо директивы после неё может стоять условие или несколько условий. Для директивы parallel это может быть, например условие if, который накладывает условие на распараллеливание, или shared и private, которые указывают на то какие переменные следует использовать совместно и какие будут личными для каждого потока, соответственно.

В своей работе помимо shared и private использовал следующие условия:

`schedule` – указывает на то, как будет разделена работа цикла между потоками.

`collapse` – указывает количество вложенных циклов, которых необходимо учитывать при распределении работы между потоками. В документации спецификации написано, что это может улучшить производительность.

`reduction` – определяет операцию редукции над переменными. Для каждой указанной переменной для каждого потока будет создана приватная переменная, а в конце блока будет посчитано финальное значение в зависимости от нужд. Пример: `reduction(min:a)`. В конце блока в a будет лежать минимальное значение между всеми потоками.

`default` – указывает на модификатор переменных по умолчанию. Я использую `default(none)`, потому что хочу явно задать какие переменные у меня будут в моих блоках (явное лучше, чем неявное).

Описание работы

Я выбрал вариант 9 - Нормализация яркости изображения.

Условие:

Автоматическая коррекция яркости изображения в цветовом пространстве YCbCr.601.

Значение пикселей изображения находится в диапазоне [0; 255]. Изображение может иметь плохую контрастность: используется не весь диапазон значений, а только его часть. Например, если самые тёмные места изображения имеют значение 20.

Задание состоит в том, чтобы изменить значения пикселей таким образом, чтобы получить максимальную контрастность (диапазон значений [0; 255]) и при этом не изменить цветность (оттенок). Этого можно достигнуть регулировкой контрастности в канале яркости Y цветового пространства YCbCr (601 в PC диапазоне: [0; 255]).

Важно: согласно стандарту PNM изображения хранятся в цветовом пространстве RGB.

<параметры_алгоритма> = <имя_входного_файла>

Входной файл содержит данные в формате PPM (P6).

В качестве выходного файла будет новое изображение в формате PPM (P6). Имя выходного файла для данного варианта гарантировано будет указано в аргументах.

Алгоритм:

1) Читаем ppm изображение (проверяем что оно является P6 8 битным изображением (по ТЗ))
2) Находим Y канал для каждого пикселя из значений RGB
3) Считаем минимум и максимум значения Y для каждого пикселя
4) Переносим диапазон значений Y с диапазона [YMin, YMax] на диапазон [0;255]
5) Находим новые значения R, G и B для каждого пикселя используя новое значение Y.
6) Запишем полученную картинку в файл.

Результат работы и замеры скорости

Исходное изображение размера 3840 x 2160 пикселей.

Рисунок 2 - Исходное изображение

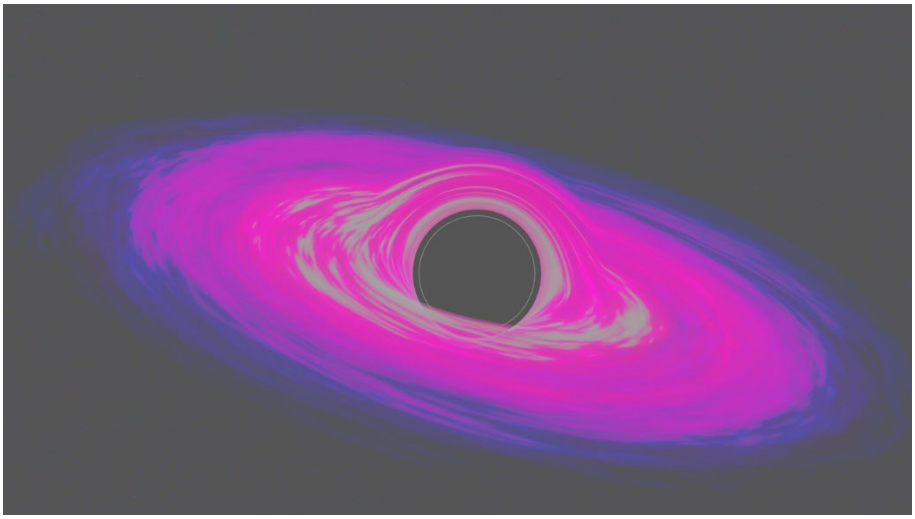

Рисунок 3 - Изображение с плохим контрастом

Рисунок 4 - Результат работы алгоритма

Для замера скорости я сделал 2 теста с ключевым отличием в способе задания schedule – во время исполнения (с помощью переменной окружения) или во время компиляции. На каждых параметрах программа запускалась 10 раз для получения достоверных данных. Используется 8 потоков поскольку на моём ноутбуке 4 ядра и включен Hyper threading.

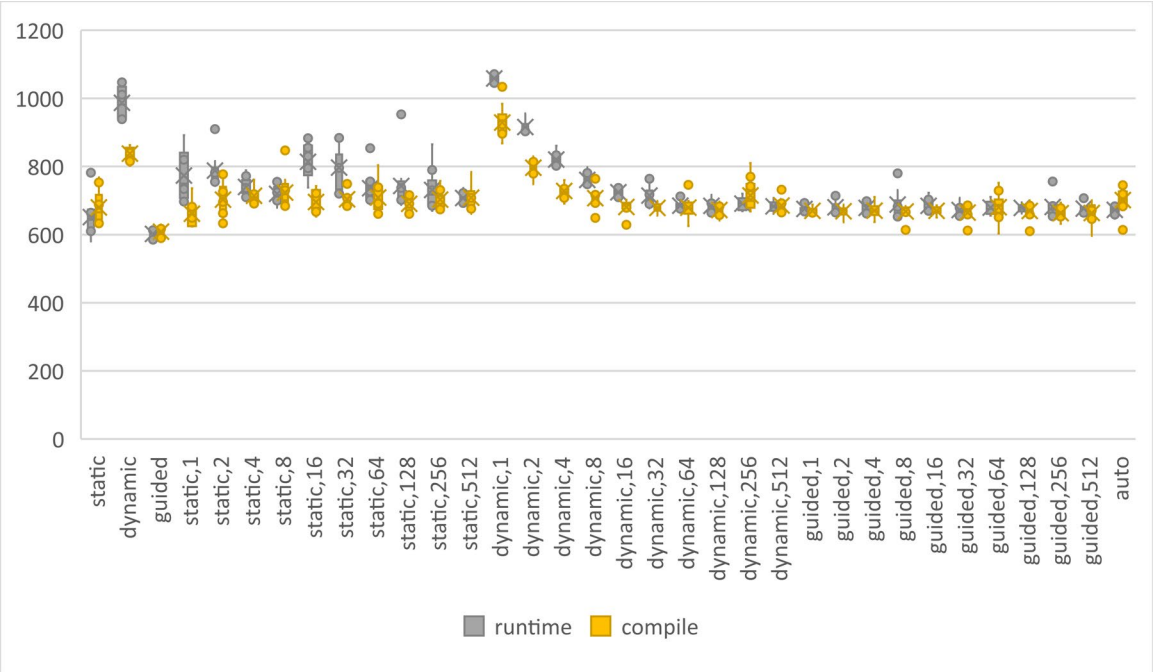

Рисунок 5 - График времени от параметра schedule

Заметно что самый быстрый результат давал schedule(guided) указанный во время компиляции. Для следующего теста буду использовать его.

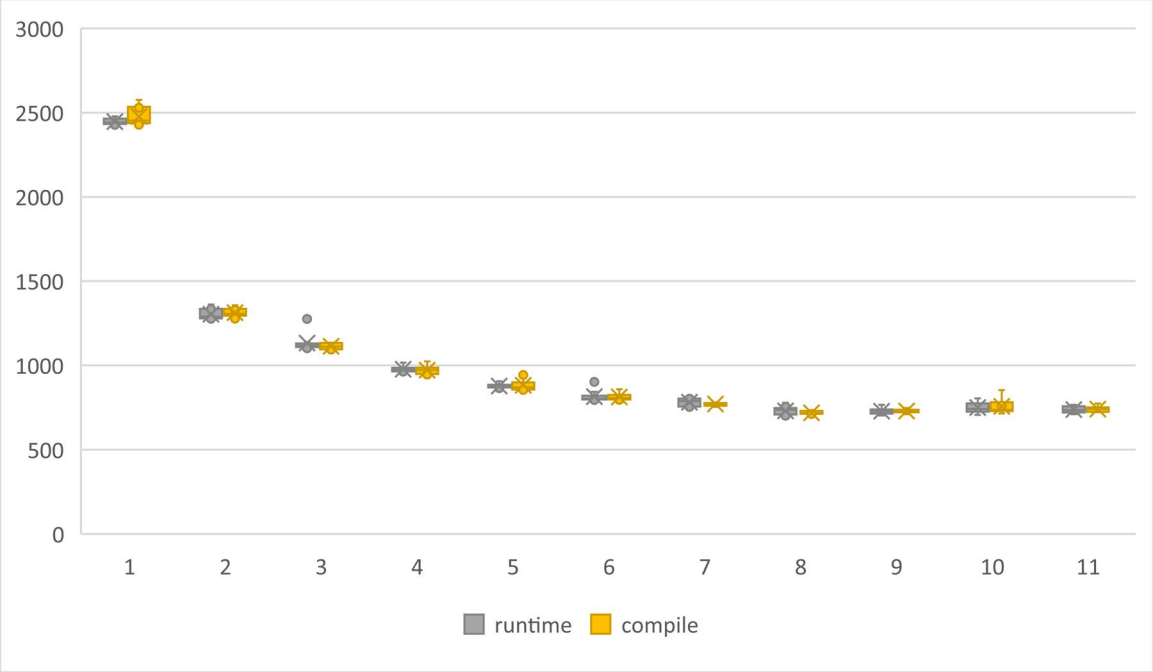

Рисунок 6 - График времени от количества потоков

Как видно после 8 потоков идёт только ухудшение. Это ожидаемо, поскольку как уже было написано выше, на моей системе 8 логических ядер.

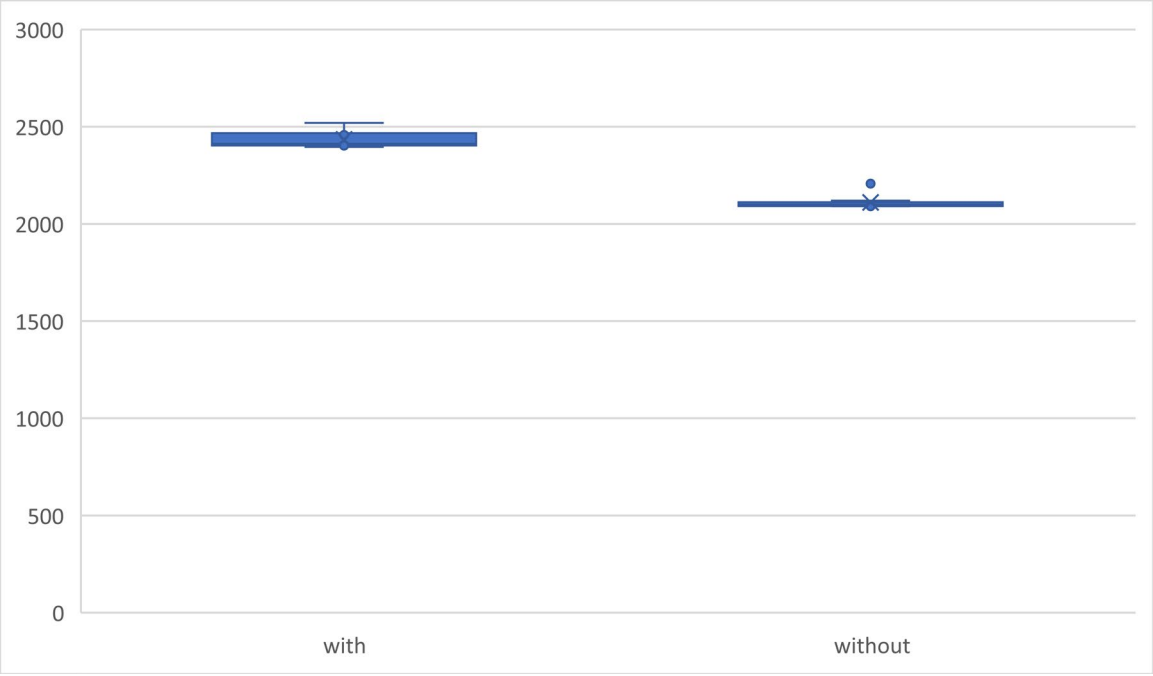

Рисунок 7 - Сравнение с влюченым и выключеным openmp

Листинг кода

Язык – C11

Компилятор – GNU gcc 10.2.0

main.c

```c
#include "ppm.h"
#include "normalize.h"
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/timeb.h>
#include <string.h>


int main(int argc, char *argv[]) {
    if (argc != 4) {
        fputs("Usage: ", stderr);
        fputs(argv[0], stderr);
        fputs(" <threads> <input> <output>\n", stderr);
        return -1;
    }
    char *endptr;
    errno = 0;
    long threads_n = strtol(argv[1], &endptr, 10);
    if (endptr != argv[1] + strlen(argv[1]) || errno == ERANGE) {
        fputs("Number of threads must be an integer\n", stderr);
        return -1;
    }
    omp_set_num_threads(threads_n);

    image *img = ppm_read_image(argv[2]);
    if (img == NULL) {
        fputs(ppm_error_message, stderr);
        fputs("\n", stderr);
        return -1;
    }

    struct timeb start, end;
    ftime(&start);
    normalize(img);
    ftime(&end);

    long long ns_start = (long long) start.millitm + start.time * 1000;
    long long ns_end = (long long) end.millitm + end.time * 1000;
    printf("Time (%ld thread(s)) - %fms\n",
           threads_n, ((float) (ns_end - ns_start)));

    if (!ppm_write_image(img, argv[3])) {
        fputs(ppm_error_message, stderr);
        fputs("\n", stderr);
        return -1;
    }
```

```
    image_destruct_image(img);
    return 0;
}
```

ppm.h

```
//
// Created by me on 20.12.2020.
//

#ifndef HW5_PPM_H
#define HW5_PPM_H

#include "image.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <errno.h>

char *ppm_error_message = NULL;

image *ppm_read_image(char *path) {
    FILE *in = fopen(path, "rb");
    if (in == NULL) {
        ppm_error_message = "Could not open image";
        return NULL;
    }
    int width, height, l_maxval;
    if (fscanf(in, "P6%d%d%d\n", &width, &height, &l_maxval) != 3) {
        ppm_error_message = "Format error";
        fclose(in);
        return NULL;
    }
    if (l_maxval > 255) {
        ppm_error_message = "Cannot work with non 8 bit images";
        fclose(in);
        return NULL;
    }
    if (l_maxval != 255) {
        ppm_error_message = "Assertion fail. Image is not from 0 to 255.";
        fclose(in);
        return NULL;
    }
    image *img = image_init_image(width, height);
    if (img == NULL) {
        ppm_error_message = "Could not create image";
        fclose(in);
        return NULL;
    }
    size_t need = sizeof(unsigned char[height][width][3]);
    size_t read = fread(img->img, 1, need + 1, in);
    if (ferror(in)) {
        ppm_error_message = "Error while reading file";
        fclose(in);
        return NULL;
    }
    if (read < need && feof(in)) {
        ppm_error_message = "Unexpected EOF";
```

```
        fclose(in);
        return NULL;
    }
    if (read > need && !feof(in)) {
        ppm_error_message = "More data after image data";
        fclose(in);
        return NULL;
    }
    fclose(in);
    //    if (l_maxval != 255) { // Not needed because all images are from 0 to
255
    //#pragma omp parallel for schedule(static) collapse(2) shared(img, height,
width, l_maxval) default(none)
    //        for (int i = 0; i < height; i++) {
    //            for (int j = 0; j < width; j++) {
    //                image_r(img, i, j) = ((unsigned int) image_r(img, i, j)) *
255 / l_maxval;
    //                image_g(img, i, j) = ((unsigned int) image_g(img, i, j)) *
255 / l_maxval;
    //                image_b(img, i, j) = ((unsigned int) image_b(img, i, j)) *
255 / l_maxval;
    //            }
    //        }
    //    }
    return img;
}

int ppm_write_image(image *img, char *p) {
    FILE *out = fopen(p, "wb");
    if (out == NULL) {
        ppm_error_message = "Cannot open file for writing";
        return 0;
    }
    fprintf(out, "P6\n%d %d\n255\n", img->width, img->height);
    fwrite(img->img, img->width * img->height * 3, 1, out);
    fclose(out);
    return 1;
}

#endif
```

normalize.h

```
//
// Created by me on 20.12.2020.
//

#ifndef HW5_NORMALIZE_H
#define HW5_NORMALIZE_H

#include "image.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#ifndef max
#define max(a, b) ((a) < (b)?(b):(a))
#endif
```

```
#ifndef min
#define min(a, b) ((a) > (b)?(b):(a))
#endif

const float BT601a = 0.299f;
const float BT601b = 0.587f;
const float BT601c = 0.114f;
const float BT601d = 1.772f;
const float BT601e = 1.402f;
const float BT601gcrcoeff = (BT601a * BT601e / BT601b);
const float BT601gcbcoeff = (BT601c * BT601d / BT601b);

void normalize(image *img) {
    // convert to ycbcr
    // find max and min values of y
    float (*ys)[img->width][img->height] = malloc(sizeof(float[img->width][img-
>height]));
    float max_y = 0, min_y = 255;
#pragma omp parallel for schedule(auto) collapse(2) default(none) shared(ys, img,
BT601a, BT601b, BT601c) reduction(min:min_y) reduction(max:max_y)
    for (int x = 0; x < img->width; x++) {
        for (int y = 0; y < img->height; y++) {
            float r = image_r(img, x, y), g = image_g(img, x, y), b = image_b(img,
x, y);
            float yy = BT601a * r + BT601b * g + BT601c * b;
            (*ys)[x][y] = yy;
            max_y = fmaxf(max_y, yy);
            min_y = fminf(min_y, yy);
        }
    }
    const float new_min = 0;
    const float new_max = 255;
    fprintf(stderr, "%f %f\n", min_y, max_y);
#pragma omp parallel for schedule(auto) collapse(2) default(none) shared(ys,
max_y, min_y, img, new_max, new_min, BT601gcrcoeff, BT601gcbcoeff, BT601d, BT601e)
    for (int x = 0; x < img->width; x++) {
        for (int y = 0; y < img->height; y++) {
            float r = image_r(img, x, y), b = image_b(img, x, y);
            // finishing converting
            float old_y = (*ys)[x][y];
            float cb = (b - old_y) / BT601d;
            float cr = (r - old_y) / BT601e;
            // normalize
            float new_y =
                    ((old_y - min_y) * (new_max - new_min) / (max_y - min_y)) +
new_min;
            // convert to rgb
            image_r(img, x, y) = min(255, max(0,
                                              lroundf(new_y + BT601e * cr)));
            image_g(img, x, y) = min(255, max(0,
                                              lroundf(new_y - BT601gcrcoeff * cr -
BT601gcbcoeff * cb)));
            image_b(img, x, y) = min(255, max(0,
                                              lroundf(new_y + BT601d * cb)));
        }
    }
    free(ys);
}
```

```
#endif //HW5_NORMALIZE_H
```

image.h

```
//
// Created by me on 20.12.2020.
//

#ifndef HW5_IMAGE_H
#define HW5_IMAGE_H

#include <stdlib.h>

typedef struct {
    unsigned int width;
    unsigned int height;
    unsigned char *img;
} image;

#define image_r(img, x, y) img->img[((x) * (img)->height + (y)) * 3 + 0]
#define image_g(img, x, y) img->img[((x) * (img)->height + (y)) * 3 + 1]
#define image_b(img, x, y) img->img[((x) * (img)->height + (y)) * 3 + 2]

image *image_init_image(unsigned int width, unsigned int height) {
    image *img = malloc(sizeof(image));
    if (img == NULL) {
        return NULL;
    }
    img->width = width;
    img->height = height;
    img->img = malloc(sizeof(unsigned char[width][height][3]));
    if (img->img == NULL) {
        return NULL;
    }
    return img;
}


void image_destruct_image(image *img) {
    free(img->img);
    free(img);
}

#endif //HW5_IMAGE_H
```