САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ

ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №5

«**OpenMP**»

Выполнил(а): Василенко Михаил Глебович

студ. гр. М3134

Санкт-Петербург

2020

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 – это интерфейс прикладного программирования для создания многопоточных приложений, для параллельных вычислительных систем с общей памятью. OpenMp состоит из набора деректив для компиляторов и библиотек специальных функций.

Программу можно разделить на два типа областей: последовательные и параллельные. Для выполнения последовательных областей используется ровно один поток (главный), в областях распараллеливания используется также ровно один поток (в последовательных областях), этот поток является единственным на протяжение всей программы. В параллельной программе в параллельных областях используется несколько параллельных потоков. Параллельные потоки могут выполняться как на разных процессорах, так и на одном процессоре вычислительной системы и конкурировать между собой за доступ к процессору.

В параллельных программах в параллельных областях выполняется несколько параллельных потоков.

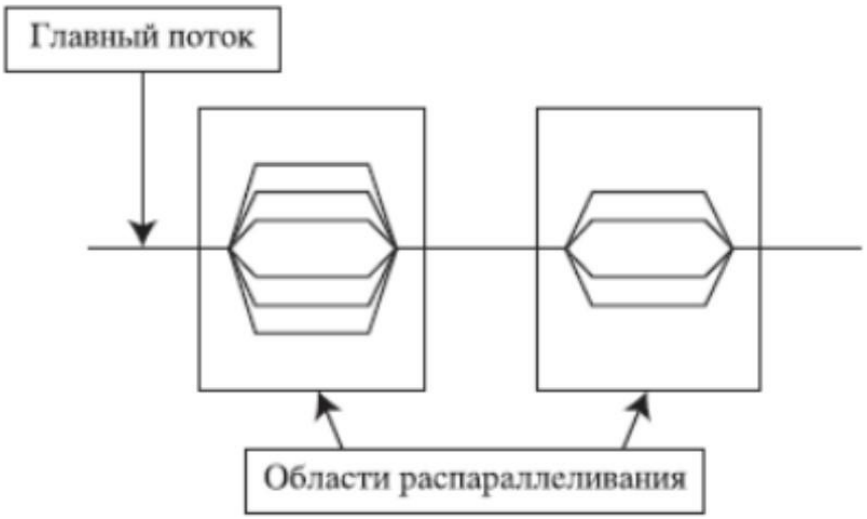

Рисунок 1 – схема параллельной программы

Параллельные потоки из одной параллельной области могут выполняться как независимо друг от друга, так и с пересылкой, получением информации друг от друга, или пересылкой, получением информации от других параллельных потоков. Очень желательно выделять такие области, распараллеливания, в которых можно организовать выполнение независимых параллельных потоков. При обращение к общим переменным в различных потоках возможны конфликты при доступе к данным. Например: рассмотрим следующий код

```
    int32_t count = 0, a[n];
#pragma parallel for
    for (int32_t i = 0; i < n; i++) {
        if (f(a[i])) {
            ++count;
        }
    }
```

Рассмотрим переменную count, она расположена в памяте, а не в регистре, тк мы хотим иметь к ней доступ из разных потоков (регистры свои у каждого потока). В строке ++count нам нужно считать переменную из

памяти, увеличить на 1 и записать обратно. Может такое произойти, что два вычислителя вошли в тело блока if одновременно, они одновременно считывают значение count (пусть 10), оба вычислителя прибавляют 1, в результате мы имеем 11 вместо 12. Могут произойти ошибки и хуже, причем они могут произойти, а могут не произойти, поэтому нельзя быть уверенным в их отсутствие, даже если протестировав программу никаких проблем не было найдено.

Решить эту проблему можно с помощью более сложной атомарной операции.

```
int32_t count = 0, a[n];
#pragma parallel for
    for (int32_t i = 0; i < n; i++) {
        if (f(a[i]))
#pragma opm critical
        {
            ++count;
        }
    }
```

Директива OpenMp состоит из “стража”, который воспринимается компилятором как комментарий, либо как начало директивы OpenMp, за которым следует сама директива и условие. Для языков C/C++ стражем является #pragma omp. Что бы они заработали нужно указать ключ компиляции -fopenmp.

Директивы, которые мне понадобились для написания программы:

For – сообщает, что тело цикла for должно быть распараллелено

Parallel – сообщает, что код должен выполняться параллельно на всех ядрах

Schedule – сообщает, как будет разделено выполнения цикла на потоки

Reduction – определяет операцию редукции для переменных. Для любой указанной переменной в каждом потока будет создана приватная переменная, в конце блока будет найдено финальное значение.

Collapse – сообщает количество вложенных циклов, которые нужно учитывать при распределении работы между потоками

Default – сообщает на модификатор переменных по умолчанию

Описание реализуемого алгоритма

Hard

Вариант 3. Автоматическая контрастность изображения

Описание задачи: Изображение может иметь плохую контрастность: используется не весь диапазон значений, а только его часть. Например, если самые тёмные точки изображения имеют значение 20, а не 0.

Задание состоит в том, чтобы изменить значения пикселей таким образом, чтобы получить максимальную контрастность: растянуть диапазон значений до [0; 255], но при этом не изменить оттенки (то есть в цветных изображениях нужно одинаково изменять каналы R, G и B).

<параметры_алгоритма> = <коэффициент>

При вычислении растяжения игнорировать некоторую долю (по количеству) самых тёмных и самых светлых точек (для RGB в каждом канале отдельно): **<коэффициент>** (диапазон значений [0.0, 0.5)). Это позволяет игнорировать шум, который незаметен глазу, но мешает автоматической настройке контрастности. Растяжение диапазона следует выполнять с насыщением, чтобы проигнорированные пиксели не вышли за границы [0; 255].

Решение

Считываем входной файл с изображением в формате .ppm, находим Y – канал для всех пикселей из значений RGB, находим минимальное и максимальное значение Y у всех пикселей, переносим диапазон значений с [YMin, YMax] на [0, 255], после находим новое значения RGB для каждого пикселя, зная новое значение Y, записываем результат в выходной файл.

Результат работы и скорость алгоритма

Рисунок 2 – изображение с плохой контрастностью

Рисунок 3 – результат работы алгоритма

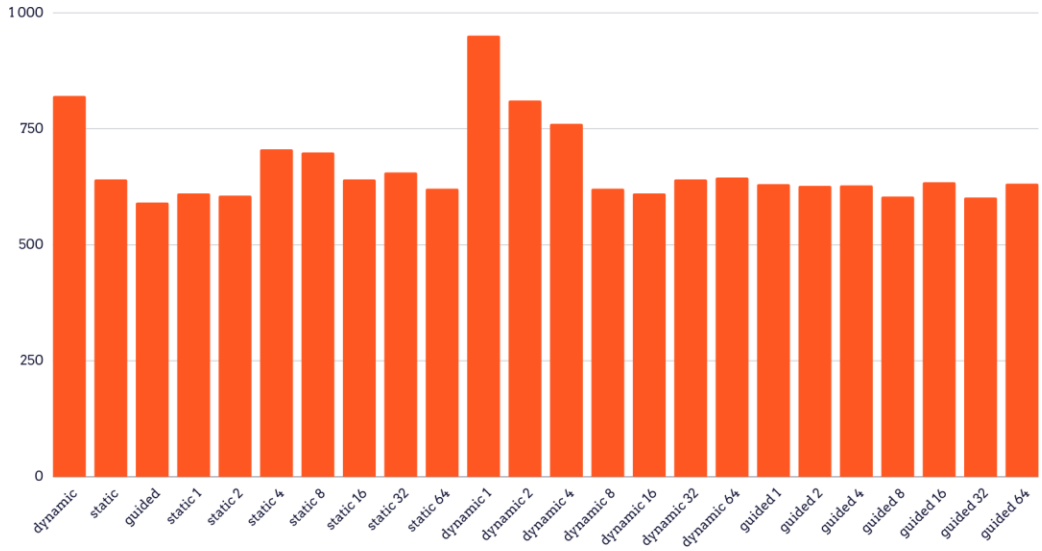

Рисунок 4 – график зависимости времени от параметра schedule

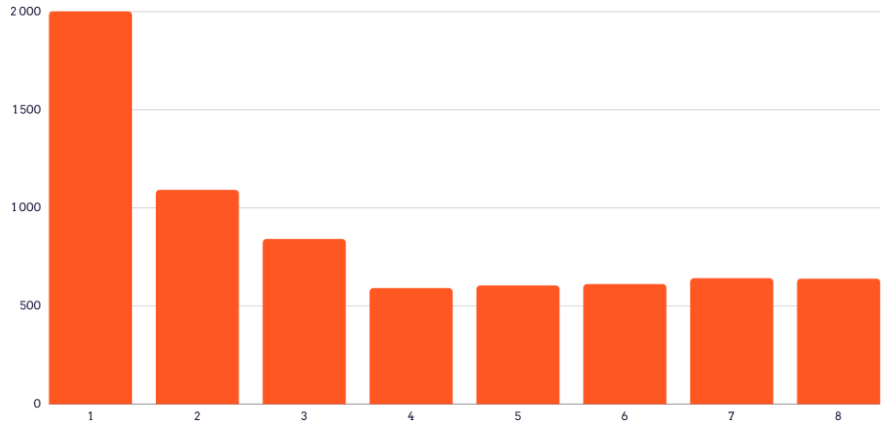

Рисунок 5 – график зависимости времени от количества тредов

Видим, что с 4 потоками скорость наиболее высокая, тк у меня 4 ядра

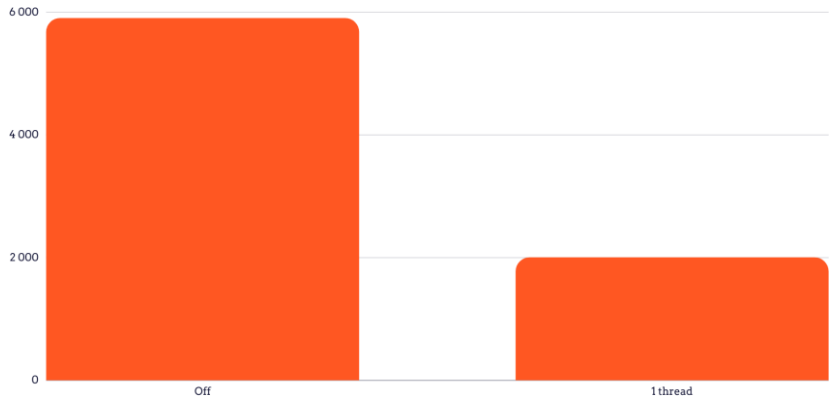

Рисунок 6 – сравнение выключенного openMp и включенного с одним потоком

Листинг кода

Язык – CNU GCC C11 5.1.0

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/timeb.h>
#include <string.h>
#include <math.h>
#include <omp.h>

typedef int32_t int32;
typedef int64_t int64;
typedef uint32_t uint32;
typedef uint64_t uint64;
typedef unsigned char uchar;
typedef float_t real;

typedef struct {
    uint32 width, altitude;
    uchar *pic;
} picture;

uchar min(int32 first, int64 second) {
    return first <= second ? first : second;
}
```

```
int64 max(int32 first, int64 second) {
    return first >= second ? first : second;
}

const real BT601Gcr = (0.299f * 1.402f / 0.587f), BT601Gcb = (0.114f *
1.772f / 0.587f);

int32 main(int32 countArgs, char *args[]) {
    if (countArgs != 5) {
        printf("wrong args: %s, %s, %s, %s, %s\n", args[0], args[1],
args[2], args[3], args[4]);
        return -1;
    }
    __unused char *endPtr;
    errno = 0;
    int64 numberOfThreads = atoi(args[1]);
    real coefficient = atof(args[4]);
    omp_set_num_threads(numberOfThreads);
    // open photo
    FILE *inputFile = fopen(args[2], "rb");
    int32 width, altitude, lMaxVal;
    fscanf(inputFile, "P6%d%d%d\n", &width, &altitude, &lMaxVal);
    picture *photo = malloc(sizeof(picture));
    if (photo == NULL) {
        return -1;
    }
    photo->width = width;
    photo->altitude = altitude;
    photo->pic = malloc(sizeof(uchar[width][altitude][3]));
    if (photo->pic == NULL) {
        return -1;
    }
    size_t need = sizeof(uchar[altitude][width][3]), read = fread(photo-
>pic, 1, need + 1, inputFile);
    fclose(inputFile);
    struct timeb start, end;
    ftime(&start);
    /*
    convert to YCbCr
    find max and min values of y
    */
    real (*ys)[photo->width][photo->altitude] = malloc(sizeof(real[photo-
>width][photo->altitude]));
    real maxY = INT32_MIN, minY = INT16_MAX;
#pragma omp parallel for schedule(auto) collapse(2) default(none)
shared(ys, photo, 0.299f, 0.587f, 0.114f) reduction(min:minY)
```

```
reduction(max:maxY)
    for (int32 row = 0; row < photo->width; row++) {
        for (int32 col = 0; col < photo->altitude; col++) {
            real r = photo->pic[((row) * (photo)->altitude + (col)) * 3 +
0],
            g = photo->pic[((row) * (photo)->altitude + (col)) * 3 + 1],
            b = photo->pic[((row) * (photo)->altitude + (col)) * 3 + 2];
            real y = 0.299f * r + 0.587f * g + 0.114f * b;
            (*ys)[row][col] = y;
            maxY = fmaxf(maxY, y);
            minY = fminf(minY, y);
        }
    }
    real nextMin = 0, nextMax = 255;
    fprintf(stderr, "%f %f\n", minY, maxY);
#pragma omp parallel for schedule(auto) collapse(2) default(none)
shared(ys, maxY, minY, photo, nextMax, nextMin, BT601Gcr, BT601Gcb,
1.772f, 1.402f)
    for (int32 row = 0; row < photo->width; row++) {
        for (int32 col = 0; col < photo->altitude; col++) {
            real r = photo->pic[((row) * (photo)->altitude + (col)) * 3 +
0],
            b = photo->pic[((row) * (photo)->altitude + (col)) * 3 + 2];
            // finishing converting
            real saveY = (*ys)[row][col];
            real c = (b - saveY) / 1.772f, cr = (r - saveY) / 1.402f;
            // normalize
            real nextY = ((saveY - minY) * (nextMax - nextMin) / (maxY -
minY)) + nextMin;
            // convert to rgb
            photo->pic[((row) * (photo)->altitude + (col)) * 3 + 0] =
                    min(255, max(0, lroundf(nextY + 1.402f * cr)));
            photo->pic[((row) * (photo)->altitude + (col)) * 3 + 1] =
                    min(255, max(0, lroundf(nextY - BT601Gcr * cr -
BT601Gcb * c)));
            photo->pic[((row) * (photo)->altitude + (col)) * 3 + 2] =
                    min(255, max(0, lroundf(nextY + 1.772f * c)));
        }
    }
    free(ys);
    ftime(&end);
    int64 startTime = (int64) start.millitm + start.time * 1000,
    endTime = (int64) end.millitm + end.time * 1000;
    printf("Time (%lld thread(s)): %g ms\\n\n", numberOfThreads, ((real)
(endTime - startTime)));
    //result
    FILE *outputFile = fopen(args[3], "wb");
```

```
    fprintf(outputFile, "P6\n%d %d\n255\n", photo->width, photo-
>altitude);
    fwrite(photo->pic, photo->width * photo->altitude * 3, 1, outputFile);
    fclose(outputFile);
    free(photo->pic);
    free(photo);
    return 0;
}
```