САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №5

«OpenMP»

Выполнил: Жимоедов Денис Евгеньевич

Номер ИСУ: 334856

студ. гр. М3134

Санкт-Петербург

2021

Цель работы: знакомство со стандартом OpenMP.

Инструментарий и требования к работе: C++17. Стандарт OpenMP 2.0.

Теоретическая часть

OpenMP – механизм написания параллельных программ для систем с общей памятью. Состоит из набора директив компилятора и библиотечных функций.

В модели с разделяемой памятью взаимодействие потоков происходит через разделяемые переменные. При неаккуратном обращении с такими переменными в программе могут возникнуть ошибки соревнования (race condition). Такое происходит из-за того, что потоки выполняются параллельно и соответственно последовательность доступа к разделяемым переменным может быть различна от одного запуска программы к другому. Для контроля ошибок соревнования работу потоков необходимо синхронизировать. Для этого используются такие примитивы синхронизации как критические секции, барьеры, атомарные операции и блокировки. Стоит отметить, что синхронизация может потребовать от программы дополнительных накладных расходов и лучше подумать, и распределить данные таким образом, чтобы количество точек синхронизации было минимизировано.

Условие **reduction** — это условие позволяет производить безопасное глобальное вычисление. Приватная копия каждой перечисленной переменной инициализируется при входе в параллельную секцию в соответствии с указанным оператором (0 для оператора +). При выходе из параллельной секции из частично вычисленных значений вычисляется результирующее и передается в основной поток.

Данное условие контролирует то, как работа будет распределяться между потоками. **schedule(тип [, размер блока])** Размер блока задает размер каждого пакета на обработку потоком (количество итераций). Тип расписания может принимать следующие значение:

static – итерации равномерно распределяются по потокам. Т.е. если в цикле 1000 итераций и 4 потока, то один поток обрабатывает все итерации с 1 по 250, второй – с 251 по 500, третий - с 501 по 750, четвертый с 751 по 1000. Если при этом задан еще и размер блока, то все итерации блоками заданного размера циклически распределяются между потоками. Статическое распределение работы эффективно, когда время выполнения итераций равно, или приблизительно равно. Если это не так, то разумно использовать следующий тип распределения работ.

dynamic – работа распределяется пакетами заданного размера (по умолчанию размер равен 1) между потоками. Как только какой-либо из потоков заканчивает обработку своей порции данных, он захватывает следующую. Стоит отметить, что при этом подходе несколько большие накладные расходы, но можно добиться лучшей балансировки загрузки между потоками.

guided – данный тип распределения работы аналогичен предыдущему, за тем исключением, сто размер блока изменяется динамически в зависимости от того, сколько необработанных итераций осталось. Размер блока постепенно уменьшается вплоть до указанного значения. При таком подходе можно достичь хорошей балансировки при меньших накладных расходах.

runtime – тип распределения определяется в момент выполнения программы. Это удобно в экспериментальных целях для выбора оптимального значения типа и размера блока.

Практическая часть

Для начала программа записывает значения в одномерный массив (можно было использовать трехмерную матрицу, но так не очень удобно параллелиться). После чего надо найти максимум и минимум (учитывая коэффициент) среди всех каналов впоследствии чего пересчитать каждый бит массива. И записать матрицу в файл. Разберем каждый основной шаг более подробно.

1) Нахождение максимума и минимума. Используем цифровую сортировку и подсчитаем на каком бите мы перейдем кол-во нужных нам => мы найдем наш минимум и максимум с коэффициентом. Цифровую сортировку можно распараллелить, но аккуратно! Так как мы можем обратиться одновременно к одной ячейке с помощью функции OpenMP reduction избавимся от этой проблемы. Так как иногда нужно найти максимум и минимум по нескольким каналом для этого мы распараллелим еще и это.
2) Пересчет. Все значения не зависят друг от друга, значит можно легко и просто распараллелить все изображение

   Проведем исследование параметров OpenMP – это кол-во потоков, тип, размер блока. Время указано в мс. Тестировалось на изображение с размером 11333*8500 RGB.

   1) Разное кол-во потоков, тип static, размер блока = 2048 (см. таблицу №1)
   2) 8 потоков, разные типы, разные размеры блоков (см. таблицу №2)

Таблица № – 1

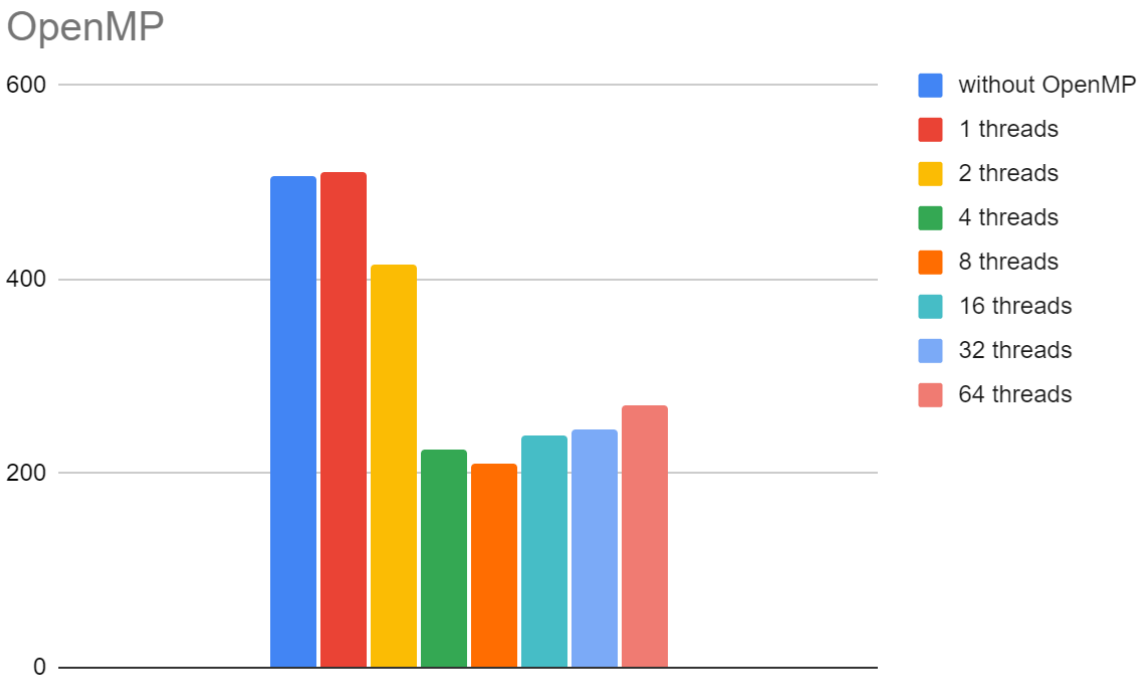
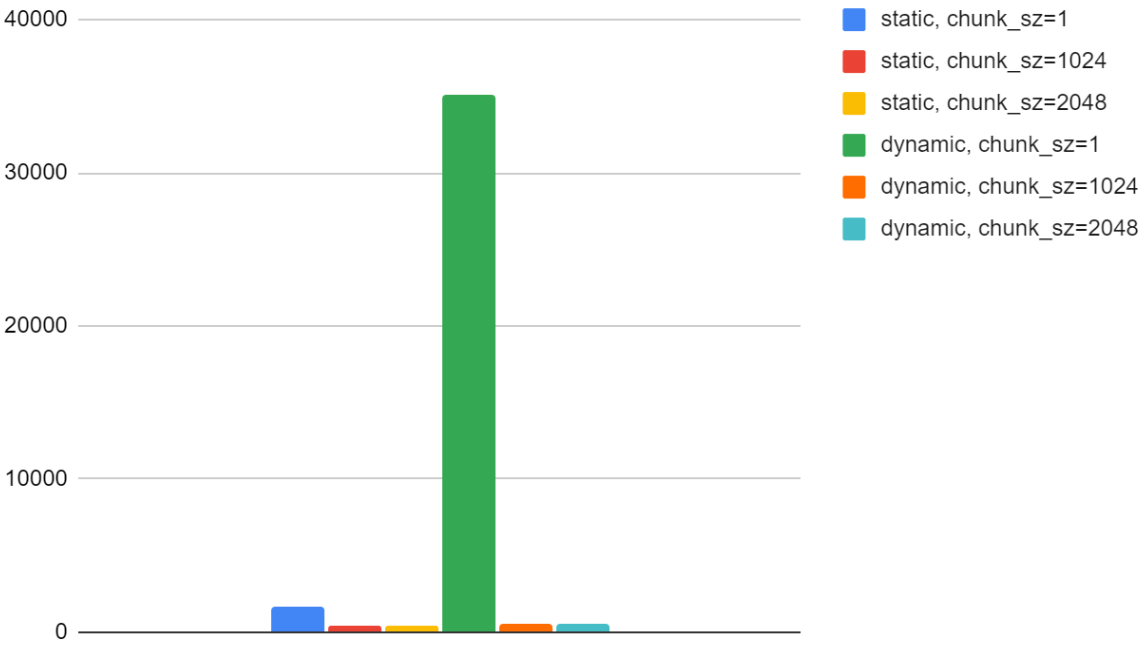


Таблица № – 2



Можно сделать вывод что после определенного кол-ва потоков не ускоряет работу программу, а иногда даже замедляется, это обусловлено тем, что мы

делаем больше, чем можем дать. По второму графику можем сделать вывод что static и 2048 подходит больше, но эти все параметры тестировались на одном примере. В ином случае может выйти хуже.

Листинг

main.cpp

```
#include <fstream>
#include <string>
#include <utility>
#include "omp.h"
#pragma GCC optimize("O3")
const long long chank_sz = 2048;
unsigned char refactorPixel(double value, double minimum, double maximum){
    if(value>=maximum){
        return 255;
    }
    else if(value<minimum){
        return 0;
    }
    else return (value-minimum)/(maximum-minimum)*255;
}
std::pair<unsigned char , unsigned char > k_static(short layer, int height, int
width, double k, unsigned char* matrix, short layers, unsigned long long sz_arr){
    double pixels_cut = height*width*k;
    int pixels_counter[256] = {0};
    unsigned char ansMin;
    unsigned char ansMax;
    long long counter = 0;
    #pragma omp parallel for schedule(static) reduction(+: pixels_counter)
    for (unsigned long long i = 0; i < sz_arr; i+=layers){
        pixels_counter[matrix[i+layer]]++;
```

```
    }
    for(int i = 0 ; i<=255; i++){ // parallel is bad
        counter+=pixels_counter[i];
        if(counter>pixels_cut){
            ansMin = i;
            break;
        }
    }
    counter = 0;
    for(int i = 255 ; i>=0; i--){ // parallel is bad
        counter+=pixels_counter[i];
        if(counter>pixels_cut){
            ansMax = i;
            break;
        }
    }
    delete[] pixels_counter;
    return std::make_pair(ansMin, ansMax);
}
int main(int argc, char *argv[]) {
    if(argc!=5){
        printf("Error. Need 5 argument");
        return 0;
    }
    int width, height, colorPolitre;
    short layers=0;
    try{
        std::ifstream filein;
        std::ofstream fileout;
        std::string inputFile = argv[2];
        std::string outputFile = argv[3];
        std::string str;
```

```
        double coef = atof(argv[4]);
        unsigned char paramMax=0;
        unsigned char paramMin=255;
        int num_threads = atof(argv[1]);
        delete argv;
        try{
            if(num_threads==0) num_threads = omp_get_max_threads();
            omp_set_num_threads(num_threads);
            try{
                filein.open(inputFile, std::fstream::in | std::fstream::binary);
                fileout.open(outputFile, std::fstream::out |
std::fstream::binary);
                filein >> str;
                fileout << str << '\n';
                //cout << str << '\n';
                if(str=="P6"){
                    layers = 3;
                } else if (str=="P5"){
                    layers = 1;
                } else{
                    printf("file isn't correct for our task");
                    return 0;
                }
                filein >> width;
                fileout << width << '\n';
                //std::cout << "WIDTH : " << width << '\n';
                filein >> height;
                fileout << height << '\n';
                //std::cout << "HEIGHT : " << height << '\n';
                filein >> colorPolitre;
                fileout << colorPolitre << '\n';
                if(colorPolitre!=255){
                    printf("Problem with file. Tz hate this!");
```

```
                    return 0;
                }
                //std::cout << "COLOUR COUNTER : " << colorPolitre << '\n';
                unsigned long long const sz_arr = width*height*layers;
                unsigned char *matrix = new unsigned char[sz_arr];
                filein.read((char*) matrix, 1);
                filein.read((char*) matrix, sz_arr);
                filein.close();

                double start_time;
                start_time = omp_get_wtime();

                #pragma omp parallel for schedule(static) reduction(max: paramMax)
reduction(min: paramMin)
                for(short layer = 0; layer < layers; layer++){
                    std::pair<unsigned char, unsigned char> minimumAndMaximum =
k_static(layer, height, width, coef, matrix, layers, sz_arr);
                    //std::cout << "layer of " << layer << " have max is " <<
int(minimumAndMaximum.second) << " AND min is " << int(minimumAndMaximum.first) <<
'\n';
                    paramMax = std::max(minimumAndMaximum.second, paramMax);
                    paramMin = std::min(minimumAndMaximum.first, paramMin);
                }

                //std::cout << "layers active : " << layers << '\n';
                //std::cout << "Colour min coef is " << int(paramMin) << '\n';
                //std::cout << "Colour max coef is " << int(paramMax) << '\n';
                unsigned char *convert = new unsigned char[256];
                for(long long i = 0; i < 256; i++){
                    convert[i] = refactorPixel(i, paramMin, paramMax);
                }
                #pragma omp parallel for schedule(static)
                for(long long i = 0 ; i < sz_arr; i++){
                    matrix[i] = convert[matrix[i]];
```

```
                }
                delete[] convert;
                double end_time;
                end_time = omp_get_wtime();
                printf("Time (%i thread(s)): %g ms\n", num_threads, (end_time -
start_time)*1000);
                fileout.write((char*)matrix, sz_arr);
                delete[] matrix;
                fileout.close();
            } catch(int e){
                printf("Can't open file or not found file or problem with img");
                return 0;
            }
        } catch(int e){
            printf("Problem with threads");
            return 0;
        }
    } catch(int e){
        printf("Problem with arguments");
        return 0;
    }
}
```