Лабораторная работа №4	М3138	2022
OpenMP	Бекаревич Анна Павловна	

Цель работы: знакомство с основами многопоточного программирования.

Инструментарий и требования к работе: C, GCC 6.3.0 mingw.

Описание: Изучить конструкции OpenMP для распараллеливания вычислений. Написать программу, осуществляющую пороговую фильтрацию изображения методом Оцу.

Описание конструкций OpenMP для распараллеливания команд

OpenMP – стандарт для распараллеливания программ. Директивы стандарта OpenMP позволяют сделать так, что бы при заходе в параллельный регион мастер-поток создавал группу потоков (это происходит благодаря fork), при выходе из этого региона осуществляется синхронизация (fork-join параллелизм), но поток остается открытым для последующего использования с целью экономии времени.

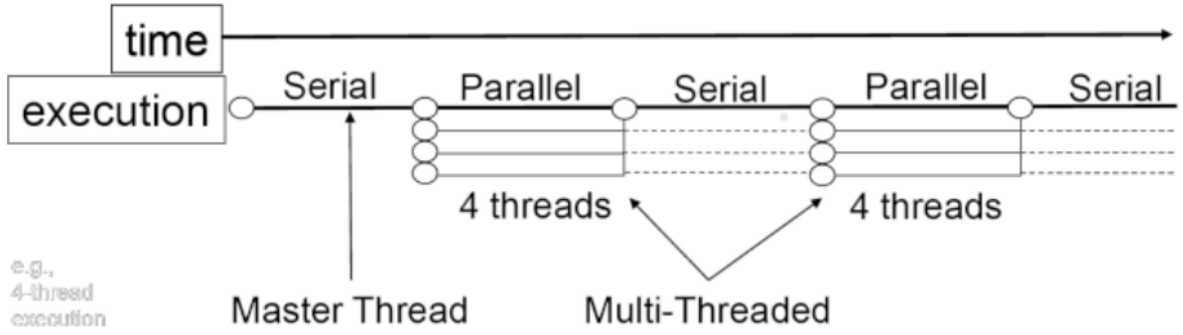

Рис 1. - работа OpenMP

Конструкция #pragma используется для задания дополнительных указаний компилятору. Использование в ней специальной ключевой директивы omp

указывает на то, что команды относятся к OpenMP. Директива parallel создает параллельный регион для следующего за ней структурированного блока, указывает, что структурный блок кода должен быть выполнен параллельно в несколько потоков. У неё есть много различных директив, которые отвечают за её поведение. Для распараллеливания с помощью директивы for необходим предварительно активировать параллельный регион. Директива for распараллеливает выполнение операций цикла, каждый поток выполняет некоторую часть от общего количества итераций. За регуляцию распределения итераций цикла отвечает параметром schedule. Он может принимать значения:
1. static – итерации статически разделяются между потоками, делясь на блоки по chunk(по дефолту итерации делят равномерно и непрерывно) итераций.

2. dynamic – по дефолту chunk=1, итерационные блоков динамически распределяются.

3. guided – по дефолту chunk=1 (определяет минимальный размер блока размер итерационного блока) и уменьшается экспоненциально при каждом распределении.

4. runtime – правило распределения определяется переменной окружения

Данные для параллельных регионов могут быть общими или частными (shared/private). Общие данные доступны всем потокам, а частные могут изменяться только одним потоком, так как принадлежат только ему. Если объект создается вне параллельного региона, то она общая, иначе частная.

Описание работы написанного кода

Все конструкции #pragma OpenMp, которые я использовала, просто вставляются в код там, где нужно создать параллельный регион, я параллелила итерации циклов, поэтому вставляла их перед началом очередного цикла, при этом там, где перед началом распараллеливания были многомерные циклы, например трехмерные, я переделала их в одномерные. Мне пришлось решать конфликт, возникший из-за того что

разные потоки обращались к одной ячейке памяти, поэтому сделала двумерный массив, где поток писал в ячейку со своим номером (для каждого значения добавлена дополнительная размерность по числу потоков). В результате все эти данные суммировались по всем потокам для каждого значения. Это оптимально по обращению к памяти - массив лежит в памяти последовательно и попадает в кэш и при этом мы не теряем во времени из-за конфликта с данными.

Для отслеживания времени работы и построения по нему графиков использовала omp_get_wtime – это функция, возвращающая значение с плавающей запятой двойной точности, равное часовому интервалу прошедшего работы в секундах, начиная с некоторого времени. Ещё мне понадобилась функция omp_get_max_threads, возвращающая целое число, которое гарантированно должно быть не меньше числа потоков, которые будут использоваться для формирования команды, omp_set_num_threads - функция, задающая количество потоков по умолчанию, используемых для последующих параллельных регионов, omp_get_thread_num - функция, возвращающая номер потока в диапазоне от 0 до omp_get_num_threads() -1 включительно в своей команде потока, в котором выполнялась функция, при этом главный поток - 0.

Для задания schedule есть #define OMP_FOR_MODE.

Для получения статистических данных написаны bash-скрипты:

check.sh - компилирует и запускает 10 раз с заданными параметрами

get_data.sh - собирает данные для трех типов schedule и разного числа потоков

get_data2.sh - собирает данные по разным schedule

get_data3.sh - собирает данные при числе потоков -1 и без -fopenmp.

Результат работы написанной программы с указанием процессора, на котором производилось тестирование.

Процессор - Процессор Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz, 2496 МГц, ядер: 4, логических процессоров: 8

Было дано изображение in.pgm

Рис 2. - данное изображение

Было получено изображение out.pgm

Рис 3. - полученное изображение

Значения порогов были:

f0 - 77

f1 - 130

f2 - 187

Экспериментальная часть (hard)

Рассмотрим время работы при различных schedule и числе потоков на следующих трех графиках (По горизонтали - число потоков и режим, по вертикали - время):

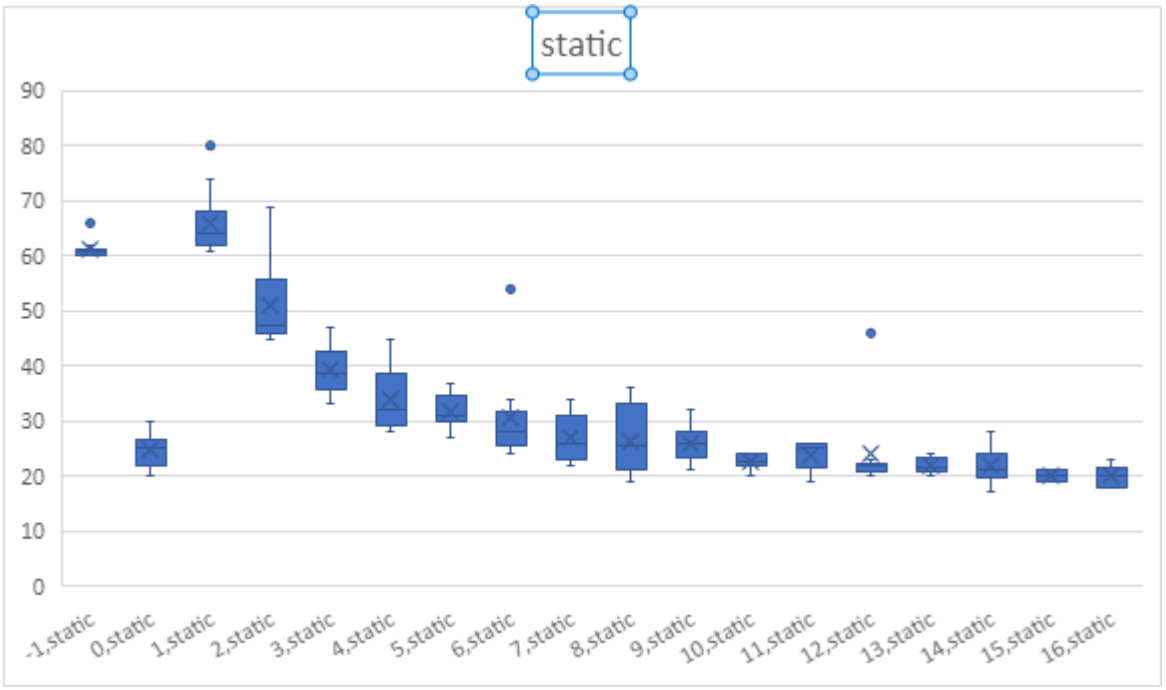

Диаграмма 1

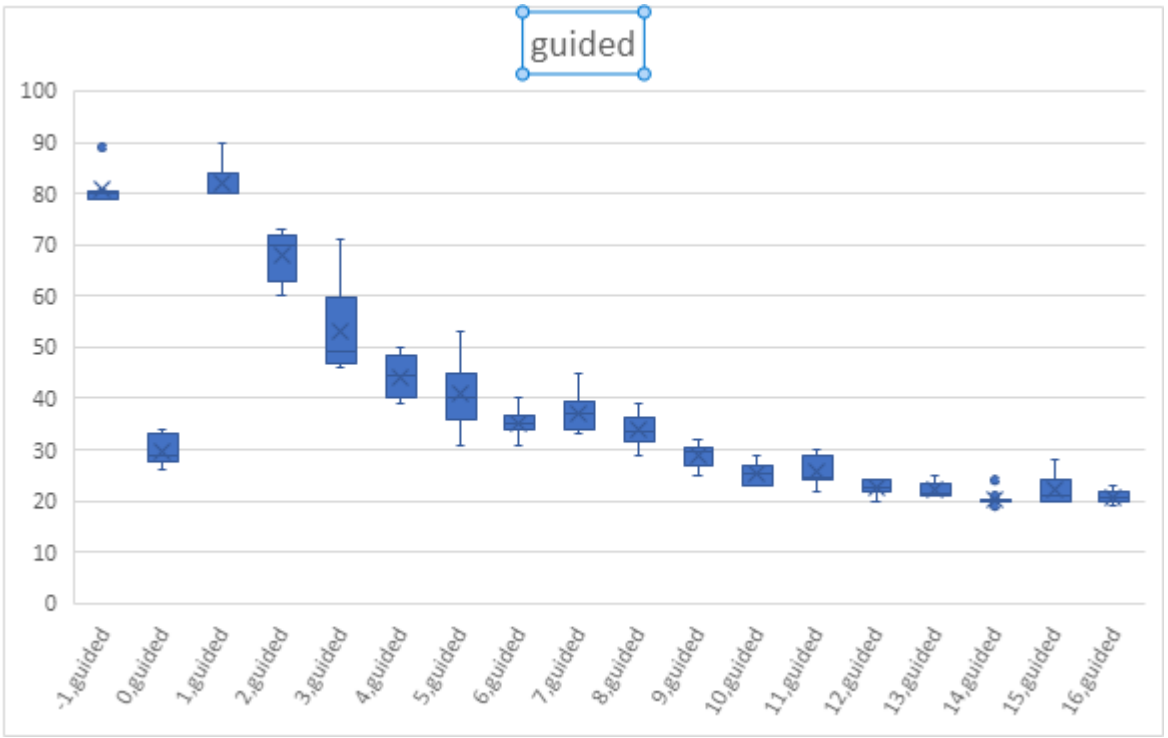

Диаграмма 2

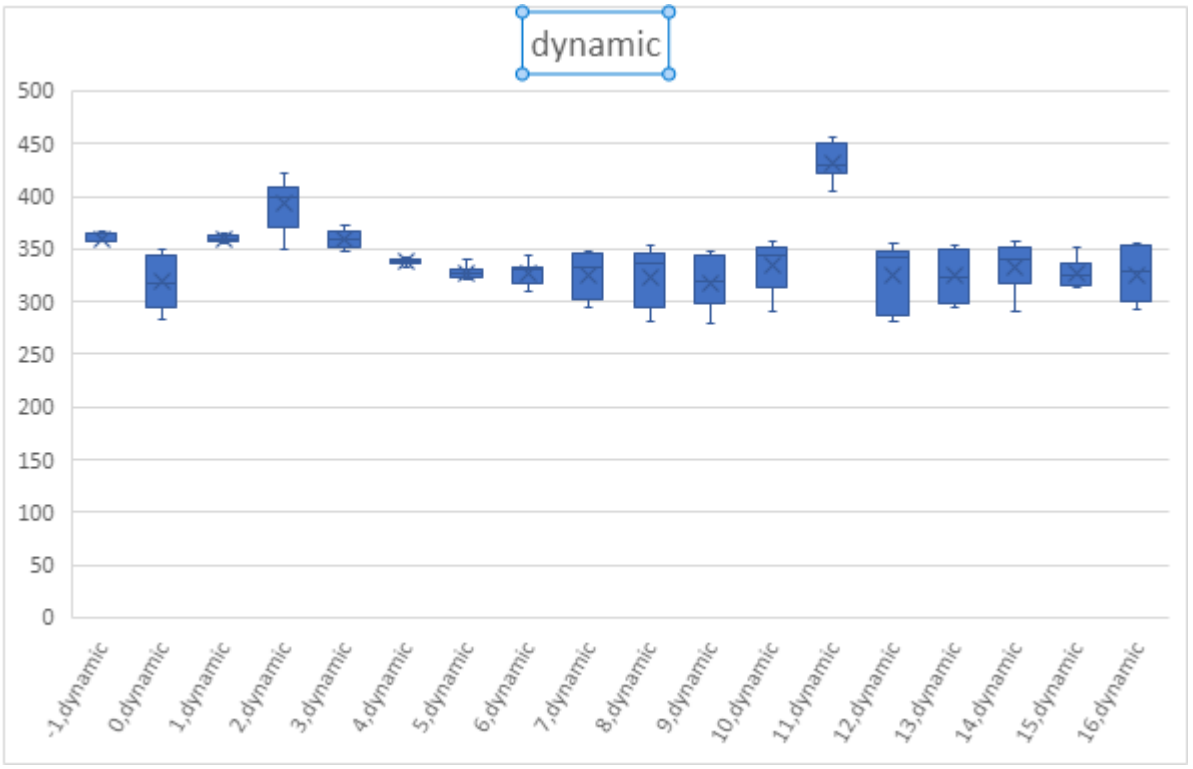

Диаграмма 3

Как видно из графиков, оптимальное значение времени начинаем получать с 8 потоков и далее изменения незначительны. Выглядит логично для системы с 8ю логическими ядрами.

Далее зафиксируем 8 потоков и рассмотрим различные значения schedule на следующих трех графиках (по горизонтали - режим, по вертикали - время):

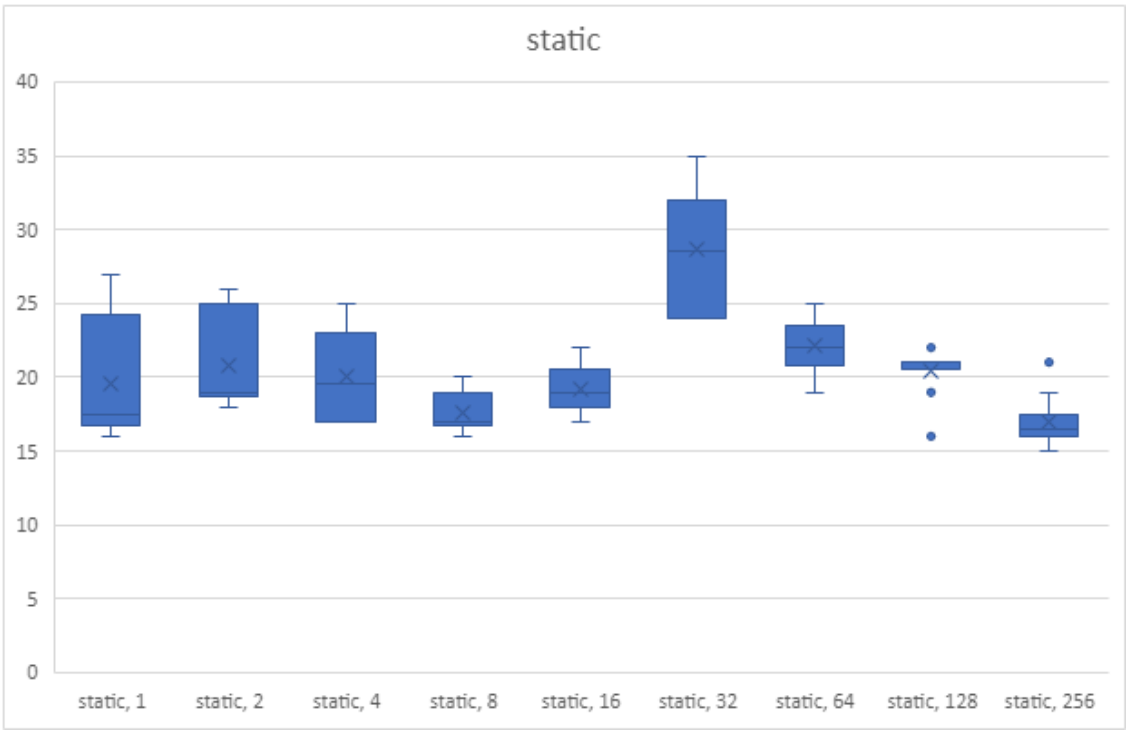

Диаграмма 4

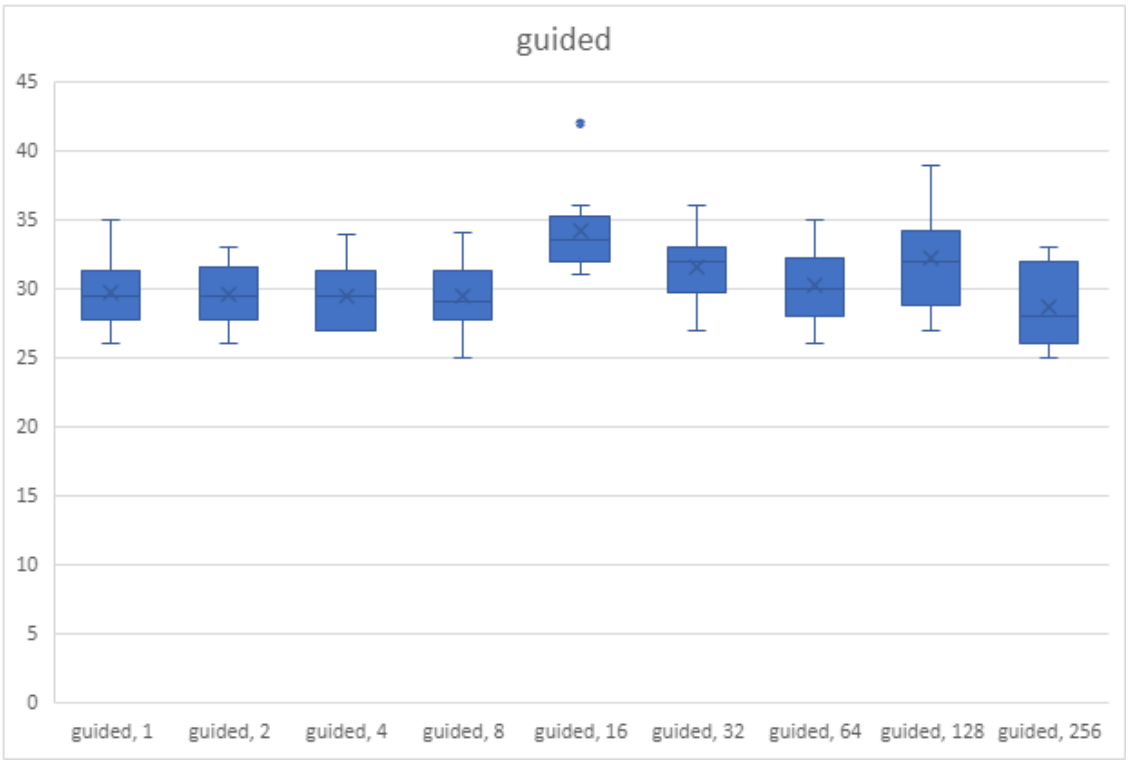

Диаграмма 5

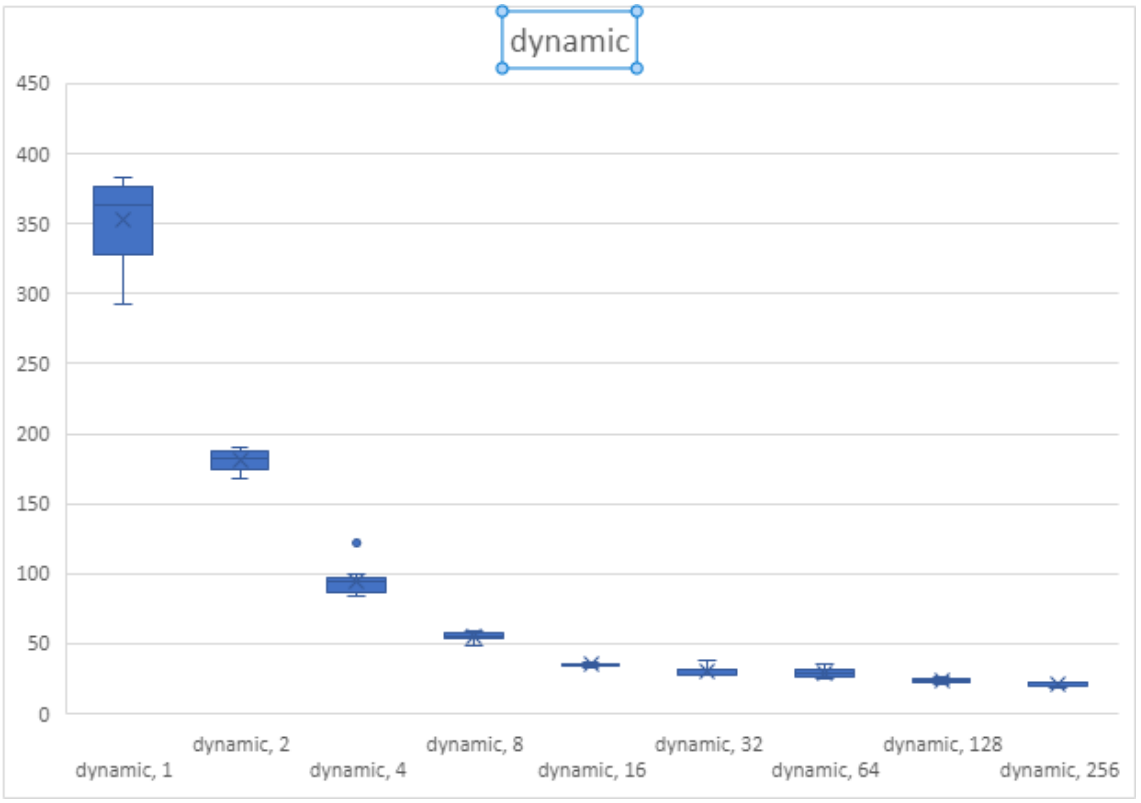

Диаграмма 6

Отсюда делаем вывод что оптимальный режим - schedule с достаточным разбиением

Исследуем разницу между флагом -1 и компиляцией без openmp (по горизонтали - режим, по вертикали - время), из диаграммы 7 видно, что компиляция без openmp выигрывает по времени.

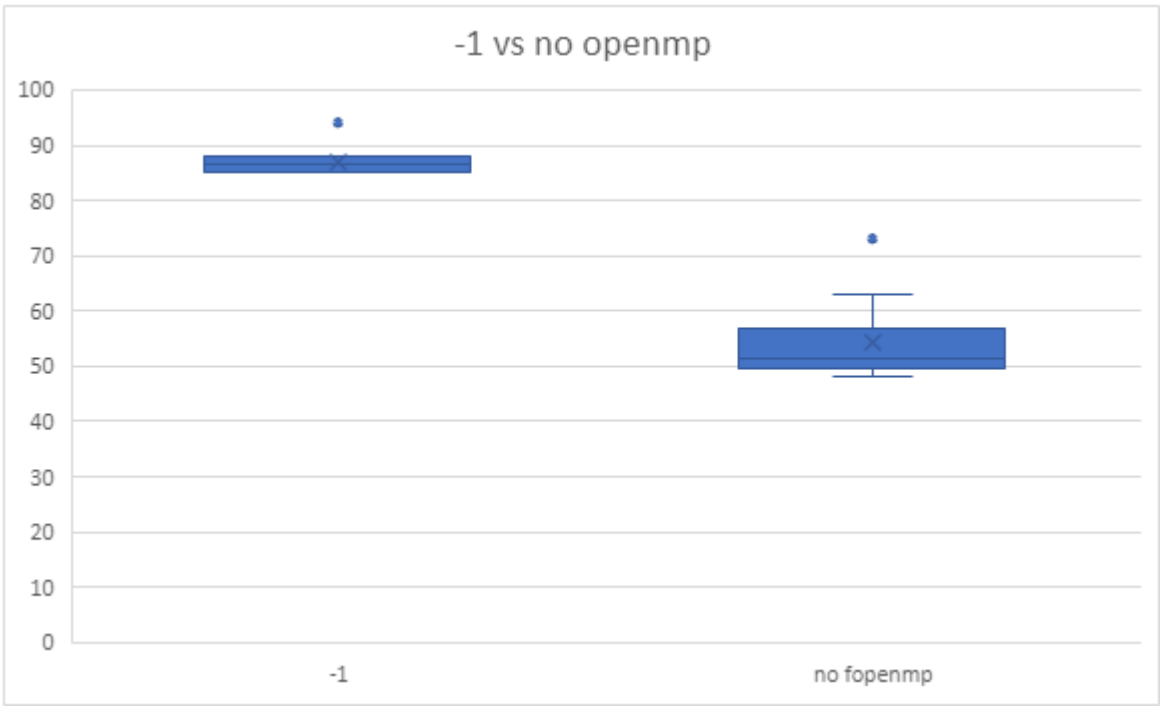

Диаграмма 7

Сравнив и проанализировав полученные данные о времени работы по данным диаграммам, я выбрала оптимальное решение для оптимизации своей программы.

Список источников

18. OpenMp: Директивы для распределения вычислений внутри параллельной области: Директива for. (studfile.net)

3. функции библиотеки времени выполнения | Microsoft Learn

Результаты поиска по запросу «[параллельное программирование]» / Хабр (habr.com)

Переменные окружения · OpenMP (gitbooks.io)

Листинг кода.

hard.c

```
#include <stdio.h>
#include<stdlib.h>
#include <omp.h>
#include <string.h>
#ifdef _WIN32
   #include <sys\timeb.h>
#else
   #include <sys/time.h>
#endif
typedef unsigned char byte;

#ifndef OMP_FOR_MODE
   #define OMP_FOR_MODE dynamic, 96
#endif
byte * image;
int width, height;
byte f0, f1, f2;
long long n;

long long histogram[256];
long long * histogram_calc[256];
double image_p[256];

double q_prefix[256];
double y_prefix[256];

int max_threads;
int thread_count;

void read_image(char * path){
   FILE * file = fopen(path, "rb");
   if(file == NULL){
      printf("error opening file!\n");
   }
   fscanf(file, "P5\n%d %d\n255\n", &width, &height);
   n = ((long long)width) * height;
   image = (byte*)malloc(sizeof(byte) * n);
   fread(image, sizeof(byte), n, file);
   fclose(file);
}

void write_image(char * path){
```

```
    FILE * file = fopen(path, "wb");
    if(file == NULL){
        printf("error opening file!\n");
    }
    fprintf(file, "P5\n%d %d\n255\n", width, height);
    fwrite(image, sizeof(byte), n, file);
    free(image);
    fclose(file);
}

byte convert_pixel(byte v){
    if(v <= f0){
        return 0;
    }else if(v <= f1){
        return 84;
    }else if(v <= f2){
        return 170;
    }else{
        return 255;
    }
}

void convert_image(){
    #pragma omp parallel for schedule(OMP_FOR_MODE) if(thread_count != -1)
    for(long long i = 0; i < n;i++){
        image[i] = convert_pixel(image[i]);
    }
}

void caclulate_histogram(){
    #pragma omp parallel for schedule(OMP_FOR_MODE) if(thread_count != -1)
    for(long long i = 0; i < n;i++){
        #ifdef _OPENMP
            histogram_calc[image[i]][omp_get_thread_num()]++;
        #else
            histogram_calc[image[i]][0]++;
        #endif
    }
    #pragma omp parallel for schedule(OMP_FOR_MODE) if(thread_count != -1)
    for(int i = 0; i < 256;i++){
        for(int g = 0; g < max_threads;g++){
            histogram[i] += histogram_calc[i][g];
        }
    }
}

double calculate_q(int from, int to){
    return q_prefix[to] - (from == 0 ? 0 : q_prefix[from - 1]);
```

```
}

double calculate_y(int from, int to, double q){
    return (y_prefix[to] - (from == 0 ? 0 : y_prefix[from - 1])) / q;
}

double caclulate_dispersion(int p0, int p1, int p2){
    double q1 = calculate_q(0, p0);
    double q2 = calculate_q(p0 + 1, p1);
    double q3 = calculate_q(p1 + 1, p2);
    double q4 = calculate_q(p2 + 1, 255);

    double y1 = calculate_y(0, p0, q1);
    double y2 = calculate_y(p0 + 1, p1, q2);
    double y3 = calculate_y(p1 + 1, p2, q3);
    double y4 = calculate_y(p2 + 1, 255, q4);

    double y = q1 * y1 + q2 * y2 + q3 * y3 + q4 * y4;

    return q1 * (y1 - y)*(y1 - y) + q2 * (y2 - y)*(y2 - y) + q3 * (y3 - y)*(y3 - y) + q4 * (y4 - y)*(y4 -
y);
}

void precalc(){
    for(int i = 0; i < 256;i++){
        image_p[i] = ((double)histogram[i]) / n;
        if(i == 0){
            q_prefix[i] = image_p[i];
            y_prefix[i] = i * image_p[i];
        }else{
            q_prefix[i] = q_prefix[i - 1] + image_p[i];
            y_prefix[i] =  y_prefix[i - 1] + i * image_p[i];
        }
    }
}
void calculate_f(){
    precalc();
    double mval[max_threads];
    for(int i =0; i < max_threads; i++) mval[i] = -1;
    int t0[max_threads], t1[max_threads], t2[max_threads];
    #pragma omp parallel for schedule(OMP_FOR_MODE) if(thread_count != -1)
    for(int p = 0; p < 256*256*256;p++){
        int p0 = p / (256 * 256);
        int p1 = (p / 256) % 256;
        int p2 = p % 256;
        if(p0 >= p1 || p1 >= p2){
            continue;
        }
```

```
        if(calculate_q(0, p0) == 0 || calculate_q(p0 + 1, p1) == 0 || calculate_q(p1 +1, p2) == 0 ||
calculate_q(p2 + 1, 255) == 0){
            continue;
        }
        double dispersion = caclulate_dispersion(p0, p1, p2);
        {
            int thread;
            #ifdef _OPENMP
                thread = omp_get_thread_num();
            #else
                thread = 0;
            #endif
            if(dispersion > mval[thread]){
                mval[thread] = dispersion;
                t0[thread] = p0;
                t1[thread] = p1;
                t2[thread] = p2;
             }
        }
    }
    int ans = 0;
    for(int i = 1; i < max_threads;i++){
        if(mval[i] > mval[ans]){
            ans = i;
        }
    }
    f0 = t0[ans];
    f1 = t1[ans];
    f2 = t2[ans];
}

int main(int argc, char ** argv){
    if(argc != 4){
        printf("Wrong arguments number");
        return 0;
    }

    sscanf(argv[1], "%d", &thread_count);
    #ifdef _OPENMP
        if(thread_count > 0){
            omp_set_num_threads(thread_count);
        }
        max_threads = omp_get_max_threads();
    #else
        max_threads = 1;
    #endif
    for(int i = 0; i < 256;i++){
        histogram_calc[i] = (long long*)malloc(sizeof(long long) * max_threads);
```

```
        memset(histogram_calc[i], 0ll, sizeof(long long) * max_threads);
    }
    read_image(argv[2]);
    double start;

    #ifdef _WIN32
        struct timeb start_t, end_t;
        ftime(&start_t);


    #else
        struct timeval t1, t2;
        double elapsedTime;
        gettimeofday(&t1, NULL);
    #endif

    #ifdef _OPENMP
        start = omp_get_wtime();
    #endif
    caclulate_histogram();
    calculate_f();

    convert_image();

    #ifdef _OPENMP
        printf("%d %d %d %g\n", f0, f1, f2, (omp_get_wtime() - start) * 1000);
    #else
        #ifdef _WIN32
            ftime(&end_t);
            double diff =  (1000.0 * (end_t.time - start_t.time)+ (end_t.millitm - start_t.millitm));
            printf("%d %d %d %lf\n", f0, f1, f2, diff);
        #else
            gettimeofday(&t2, NULL);
            elapsedTime = (t2.tv_sec - t1.tv_sec) * 1000.0;
            printf("%d %d %d %lf\n", f0, f1, f2, elapsedTime);
        #endif
    #endif
    for(int i = 0; i < 256;i++){
        free(histogram_calc[i]);
    }
    write_image(argv[3]);
    // check input

}
```

check.sh

```
gcc hard.c -o hard.exe -fopenmp -O2 -D OMP_FOR_MODE="$2"
for i in {1..10}
do
   t=$(./hard.exe $1 test_data/in.pgm out.pgm | cut -d' ' -f 4)
   echo "$2;$t"
done
```

get_data.sh

```
for i in "static" "dynamic" "guided"
do
   echo ";time" > "log1_$i.csv"
   for g in {-1..16}
   do
      res=$(./check.sh $g $i | sed 's/\./,/' | awk -v var="$g" '{print var","$0}')
      echo "$res"
   done >> "log1_$i.csv"
done
```

get_data2.sh

```
echo ";time" > "log2.csv"
for i in "static" "dynamic" "guided"
do
   for g in 1 2 4 8 16 32 64 128 256
   do
      ./check.sh 8 "$i, $g" | sed 's/\./,/'
   done
done >> "log2.csv"
```

get_data3.sh

```
echo ";time" > log3.csv
gcc hard.c -o hard.exe -fopenmp -O2
for i in {1..10}
```

```
do
    t=$(./hard.exe -1 test_data/in.pgm out.pgm | cut -d' ' -f 4 | sed 's/\./,/')
    echo "-1;$t"
done >> log3.csv
gcc hard.c -o hard.exe -O2
for i in {1..10}
do
    t=$(./hard.exe -1 test_data/in.pgm out.pgm | cut -d' ' -f 4 | sed 's/\./,/')
    echo "no fopenmp;$t"
done >> log3.csv
```