САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Дисциплина: Архитектура компьютера

Отчет

по лабораторной работе №5

**«OpenMP»**

Выполнил(а): Тюряев Илья Константинович

студ. гр. Б10

Санкт-Петербург

2021

**Цель работы:** знакомство со стандартом OpenMP.

**Инструментарий и требования к работе:**C++, стандарт OpenMP.

**Описание задачи:**

Изображение может иметь плохую контрастность: используется не весь диапазон значений, а только его часть. 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).

**Теоретическая часть:**

**OpenMP**

OpenMP — открытый стандарт для распараллеливания программ на языках Си, Си++ и Фортран. Даёт описание совокупности директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью.

OpenMP реализует параллельные вычисления с помощью многопоточности, в которой ведущий поток создаёт набор ведомых потоков, и задача распределяется между ними.

Предполагается, что потоки выполняются параллельно на машине с несколькими процессорами (количество процессоров не обязательно должно быть больше или равно количеству потоков). Количество создаваемых потоков может регулироваться как самой программой при помощи вызова библиотечных процедур, так и извне, при помощи переменных окружения.

Ключевые элементы:

* конструкции для создания потоков (директива parallel),
* конструкции распределения работы между потоками (директивы DO/for и section),
* конструкции для управления работой с данными (выражения shared и private для определения класса памяти переменных),
* конструкции для синхронизации потоков (директивы ctitical, atomic и barrier),
* процедуры библиотеки поддержки времени выполнения (например, omp\_get\_thread\_num),
* переменные окружения (например, OMP\_NUM\_THREADS).

**Алгоритм**

Нужно считать некоторое количество байт из входного файла, при помощи которых можно будет понять формат входного изображения (PGM или PPM), а также его размер.   
Далее нужно считать требуемое количество байт – по три на пиксель для PPM, и по одному на пиксель для PGM.   
Среди соответствующих байт всех пикселей найти минимальное и максимальное значение среди них (мы будем производить изменения каналов R,G,B для PPM независимо, чтобы не поменять оттенки). Это позволит определить отрезок, в котором находятся значения соответствующих байт пикселей, который мы хотим растянуть до отрезка [0,255].   
Далее нам нужно изменить значения байт каждого пикселя в соответствии с некоторым отображением из соответствующего отрезка в отрезок [0,255]. Рассмотрим необходимое преобразование для какой-либо группы байт пикселей(например, только для канала R у PPM, либо единственного канала PGM – для остальных рассуждения аналогичны). Если все значения совпадают – то считается, что максимизировать контрастность не получится из соображений мощности(отрезок – континуум, точка – конечна), поэтому применяем тождественное отображение – не меняем значения. Если же значения лежат в некотором отрезке, то возьмем биективное отображение из этого отрезка в отрезок [0,255](как бы растянув его), например, если x – точка отрезка [min,max] :

Далее остается записать в выходной файл формат изображения, размер, а также измененные значения пикселей.

**Практическая часть:**

**Описание работы кода**

* void throw\_error(std::string&& err);

функция, выводящая ошибку в стандартный поток вывода и завершающая работу программы с кодом выхода -1

(используется std::move для того, чтобы избежать лишнее копирования строки с ошибкой)

* inline uint8\_t calc\_norm(uint8\_t x, uint8\_t l, uint8\_t r);

функция, вычисляющая функцию из теоретической части  
(функция представляет из себя короткую формулу – имело смысл сделать inline для ускорения)

* с 21 по 51 строчку производим аккуратный ввод формата изображения и размера, обрабатывая ошибки.
* uint8\_t max\_r, min\_r, max\_g, min\_g, max\_b, min\_b, max\_grey, min\_grey;

переменные для сохранения минимальных и максимальных значений соответствующих байт среди всех пикселей

* int pixel\_sizeof = std::string(format) == "P6" ? 3 : 1;

требуемое количество байт для сохранения одного пикселя нашего изображения

* uint8\_t \*data = new uint8\_t[height\*width\*pixel\_sizeof];  
  readed\_bytes = fread(data, sizeof(uint8\_t), height\*width\*pixel\_sizeof, input\_filep);

создаем указатель на массив, в котором будем хранить все байты пикселей, и резервируем нужное количество памяти(произведение количества пикселей на количество байт, нужное для одного пикселя), далее считываем все изображение

* double start = (double)clock();  
  printf("Time (%i thread(s)): %g ms\n", (int)threads\_c, (((double)clock()-start)/CLOCKS\_PER\_SEC)\*1000);

сохраняем время, в которое начнет выполняться блок кода, время работы которого необходимо замерить, далее вычисляем время работы блока, как разность времени, в которое он закончил выполняться, и времени – в которое начал. Также необходимо полученное значение поделить на нужную для перевода в секунды константу и умножить на количество миллисекунд в секунде для получения результаты в миллисекундах.

* С 66 по 80 строку считаются необходимые максимумы и минимумы

#pragma omp parallel for   
reduction(min: min\_grey, min\_r, min\_g, min\_b)   
reduction(max: max\_grey, max\_r, max\_g, max\_b)   
shared(data, pixel\_sizeof)   
schedule(static)   
num\_threads(threads\_c)

Используется для распределения между потоками(их количество задали в num\_threads) вычислений в цикле for, при помощи shared  
пометили переменные, которые будут использовать одновременно несколькими потоками для корректной работы, при помощи reduction мы ускоряем вычисления максимум, а именно у нас будут вычисляться свои локальные максимумы для каждого потока, а потом будет взят глобальный среди всех полученных максимумов – такой подход помогает не конкурировать потокам за обращение к одной области памяти.

* С 82 по 92 строку применяем наше отображение ко всем значениям пикселей, тем самым максимизируя контрастность (с прагмой всё аналогично предыдущему пункту)
* FILE\* output\_filep = fopen(argv[3],"wb");

fprintf(output\_filep, "%s\n%d %d\n%d\n", format, width, height, bound);

fwrite(data, sizeof(uint8\_t), height\*width\*pixel\_sizeof, output\_filep);

записываем в выходной файл формат изображения, размер, а также измененные значения пикселей.

**Измерения**Измерение производительности кода производились на картинке размера ~54Mb с расширением PPM

Первый аргумент *schedule(kind):*

1. Вертикальная ось – время в миллисекундах   
   Горизонтальная ось – второй аргумент *schedule(chunk\_size)*
2. Вертикальная ось – время в миллисекундах   
   Горизонтальная ось – второй аргумент *schedule(chunk\_size)*
3. без OpenMP - 290.92ms

**Листинг кода:**

**main.cpp**

#include <algorithm>

#include <vector>

#include <utility>

#include <iostream>

#include <fstream>

#include <sstream>

#include <cassert>

#include <climits>

#include <omp.h>

void throw\_error(std::string &&err) {

printf("%s\n",err.c\_str());

exit(-1);

}

inline uint8\_t calc\_norm(uint8\_t x, uint8\_t l, uint8\_t r) {

return (r!=l ? (255/2) + (2\*x-r-l)\*(255/2)/(r-l) : x);

}

int main(int argc, char\* argv[]) {

if(!(4 <= argc && argc <= 5))

throw\_error("Wrong format of arguments");

if(!argv[1])

throw\_error("Wrong format of first argument (number of threads)");

size\_t threads\_c = atoi(argv[1]);

if(threads\_c <= 0)

throw\_error("Wrong format of first argument (number of threads should be positive numer)");

FILE\* input\_filep = fopen(argv[2], "rb" );

if(!input\_filep)

throw\_error("Wrong input file name");

char format[3];

size\_t readed\_bytes = fscanf(input\_filep, "%s\n", format);

if(std::string(format) != "P5" && std::string(format) != "P6")

throw\_error("Wrong input file format (PGM/PPM required)");

int width, height;

readed\_bytes = fscanf(input\_filep, "%d %d\n", &(width), &(height));

if(width < 0)

throw\_error("Wrong input file format (incorrect width)");

if(height < 0)

throw\_error("Wrong input file format (incorrect height)");

int bound;

readed\_bytes = fscanf(input\_filep, "%d\n", &(bound));

if(bound != 255)

throw\_error("Wrong input file format (bound must be 255)");

uint8\_t max\_r, min\_r, max\_g, min\_g, max\_b, min\_b, max\_grey, min\_grey;

max\_r = max\_g = max\_b = max\_grey = 0;

min\_r = min\_g = min\_b = min\_grey = UINT8\_MAX;

uint8\_t \*data;

int pixel\_sizeof = std::string(format) == "P6" ? 3 : 1;

data = new uint8\_t[height\*width\*pixel\_sizeof];

readed\_bytes = fread(data, sizeof(uint8\_t), height\*width\*pixel\_sizeof, input\_filep);

if((int)readed\_bytes != height\*width\*pixel\_sizeof)

throw\_error("Wrong input file format (not enough bytes in data section)");

double start = (double)clock();

#pragma omp parallel for reduction(min: min\_grey, min\_r, min\_g, min\_b) reduction(max: max\_grey, max\_r, max\_g, max\_b) shared(data, pixel\_sizeof) schedule(static) num\_threads(threads\_c)

for(int i=0;i<height\*width;++i) {

if(pixel\_sizeof == 1) {

max\_grey = std::max(max\_grey, data[i]);

min\_grey = std::min(min\_grey, data[i]);

}

else {

max\_r = std::max(max\_r, data[i\*pixel\_sizeof]);

min\_r = std::min(min\_r, data[i\*pixel\_sizeof]);

max\_g = std::max(max\_g, data[i\*pixel\_sizeof+1]);

min\_g = std::min(min\_g, data[i\*pixel\_sizeof+1]);

max\_b = std::max(max\_b, data[i\*pixel\_sizeof+2]);

min\_b = std::min(min\_b, data[i\*pixel\_sizeof+2]);

}

}

#pragma omp parallel for shared(data, pixel\_sizeof, max\_grey, max\_r, max\_g, max\_b, min\_r, min\_g, min\_b) schedule(static) num\_threads(threads\_c)

for(int i=0;i<height\*width;++i) {

if(pixel\_sizeof == 1) {

data[i] = calc\_norm(data[i], min\_grey, max\_grey);

}

else {

data[i\*pixel\_sizeof] = calc\_norm(data[i\*pixel\_sizeof], min\_r, max\_r);

data[i\*pixel\_sizeof+1] = calc\_norm(data[i\*pixel\_sizeof+1], min\_g, max\_g);

data[i\*pixel\_sizeof+2] = calc\_norm(data[i\*pixel\_sizeof+2], min\_b, max\_b);

}

}

printf("Time (%i thread(s)): %g ms\n", (int)threads\_c, (((double)clock()-start)/CLOCKS\_PER\_SEC)\*1000);

FILE\* output\_filep = fopen(argv[3],"wb");

fprintf(output\_filep, "%s\n%d %d\n%d\n", format, width, height, bound);

fwrite(data, sizeof(uint8\_t), height\*width\*pixel\_sizeof, output\_filep);

delete data;

fclose(input\_filep);

fclose(output\_filep);

}

**Makefile**

CXX = g++

CXXFLAGS = -fopenmp -O2 -Wall -Werror -std=c++17 -Iinclude

LDFLAGS = -fopenmp

EXE = hw5

SRCDIR = src

OBJDIR = obj

OBJECTS = $(patsubst $(SRCDIR)/%.cpp,$(OBJDIR)/%.o,$(wildcard $(SRCDIR)/\*.cpp))

all: $(EXE)

$(EXE): $(OBJECTS)

$(CXX) $(OBJECTS) -o $(EXE) $(LDFLAGS)

$(OBJDIR)/%.o: $(SRCDIR)/%.cpp | $(OBJDIR)

$(CXX) $(CXXFLAGS) -c -MMD -o $@ $<

include $(wildcard $(OBJDIR)/\*.d)

$(OBJDIR):

mkdir -p $(OBJDIR)

clean:

rm -rf $(OBJDIR) $(EXE)

.PHONY: clean all