1. Введение

Это задание связано с оптимизацией кода, интенсивно использующего память. Обработка изображений предлагает множество функций, которым может быть полезна оптимизация. В этой лабораторной работе, мы рассмотрим две операции по обработке изображений: `rotate`, которая переворачивает изображение против часовой стрелки на 90°, а также `smooth`, которая "сглаживает" или "размывает" изображение.

В этой лабораторной работе, мы рассмотрим изображение, представленное как двумерная матрица *M*, где *M(i,j)* обозначает значение *(i,j)*-пикселя матрицы *M*. Значение пикселя — это три значения красного, зелёного и синего цвета (RGB - red,green,blue). Мы рассматриваем только квадратные изображения. Пусть N обозначает количество строк (или столбцов) изображения. Строки и столбцы пронумерованы в стиле Си, от 0 до $N - 1$.

Имея это представление, операция `rotate` может быть реализована довольно просто, как комбинация следующих двух матричных операций:

- *Перемещение (Transpose)*: Для каждой пары *(i,j)*, *M(i,j)* и *M(j,i)* меняются местами.
- *Замена строк (Exchange rows)*: строка i заменяется строкой $N - 1 - i$

Эта комбинация проиллюстрирована рис. 1.

Операция `smooth` реализована с помощью замены каждого значения пикселя на среднее значение пикселей вокруг него (в окне 3x3, где центром является этот пиксель). Рассмотрим рис. 2. Значения пикселей `M2[1][1]` и `M2[N - 1][N - 1]` предоставлены ниже:

$$\texttt{M2[1][1]} = \frac{\sum_{\texttt{i}=0}^{2} \sum_{\texttt{j}=0}^{2} \texttt{M1[i][j]}}{9}$$

$$\texttt{M2[N-1][N-1]} = \frac{\sum_{\texttt{i}=N-2}^{N-1} \sum_{\texttt{j}=N-2}^{N-1} \texttt{M1[i][j]}}{4}$$

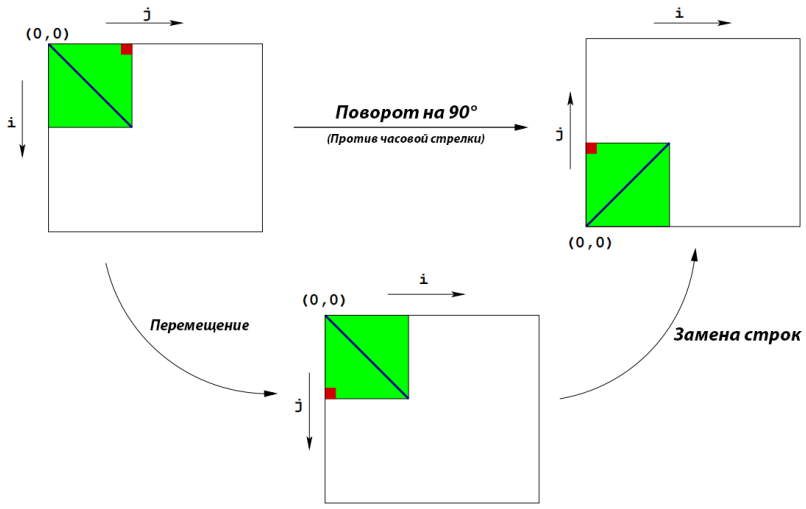

рис. 1: Поворот изображения на 90° против часовой стрелки

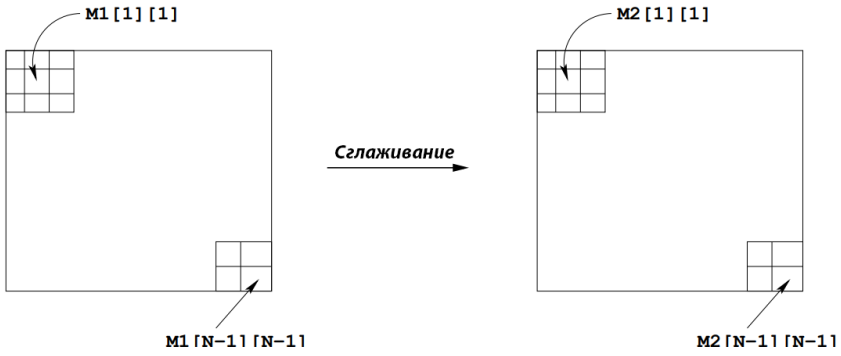

рис. 2: Сглаживание изображения

2. Инструкции по подготовке

Начните с копирования файла `perflab-handout.zip` в директорию, где вы планируете выполнять вашу работу. Распакуйте архив, и множество файлов появится в вашей директории. Единственный файыл, который вы будете изменять — это `kernels.c` и `config.h`. `driver.c` — программа-драйвер, которая позволяет вам оценить производительность ваших решений. Используйте команду `make driver`, чтобы сгенерировать код драйвера, а затем запустите его с помощью команды `./driver` Пример:

```
> make driver
```

```
> ./driver
```

Сборка может потребовать пакет 32 bit libc dev. Он имеет разные названия на разных системах Linux. Примеры установки для некоторых систем.

```
# Ubuntu: libc6-dev-i386
> sudo apt-get install libc6-dev-i386
# Red Hat: glibc-devel.i686
> sudo dnf install glibc-devel.i686
# CentOS 5.8: glibc-devel.i386
> sudo yum install glibc-devel.i386
# CentOS 6 / 7: glibc-devel.i686.
> sudo dnf install glibc-devel.i686
# SLES: glibc-devel-32bit
> zypper in glibc-devel-32bit
# Gentoo: sys-libs/glibc
> emerge -1a sys-libs/gcc
# ArchLinux: lib32-glibc
> pacman -S lib32-glibc
```

Примечания для пользователей Windows

Данную работу можно выполнить на Windows используя `Windows Subsystem for Linux`. Запустите `PowerShell` с правами администратора и настройте `WSL` для работы

Установите Windows Subsystem for Linux

```
> wsl --install
```

Запустите баш-интерпретатор

```
> bash
create account with name and password
```

Установите требуемые пакеты и пропишите путь к заголовочным файлам стандартной библиотеки.

```
> sudo apt update
> sudo apt install make build-essential libc6-dev
> export C_INCLUDE_PATH=/usr/include
```

Перейдите в директорию с файлами лабораторной работы. В данном примере, архив был распакован на рабочем столе. Рекуомендуется использовать другой путь.

```
> cd /mnt/c/Users/_username_/Desktop/perflab-handout
```

3. Общий обзор реализации

Структуры данных

Основная структура данных связана с представлением изображения. `pixel` — структура, описанная ниже:

```
typedef struct {
    unsigned short red; /* red value - R */
    unsigned short green; /* green value - G */
    unsigned short blue; /* blue value - B */
} pixel;
```

Как вы можете заметить, значения RGB имеют 16-битное представление (“16-битный цвет”). Изображение `I` представлено одномерным массивом `pixels`, где *(i,j)*-пиксель равен `I[RIDX(i,j,n)]`. Здесь `n` — это размер матрицы изображения, а `RIDX` — это макрос, определённый ниже:

```
#define RIDX(i,j,n) ((i)*(n)+(j))
```

Найдите этот код в файле `defs.h`.

Поворот

Следующая функция языка Си вычисляет результат от поворота на 90° исходного изображения `src`, а затем сохраняет его в окончательном изображении `dst`. `dim` — это измерение изображения.

```
void naive_rotate(int dim, pixel *src, pixel *dst) {
    int i, j;

    for(i=0; i < dim; i++)
        for(j=0; j < dim; j++)
            dst[RIDX(dim-1-j,i,dim)] = src[RIDX(i,j,dim)];
    return;
}
```

Вышеуказанный код сканирует строки исходной матрицы изображения, копируя столбцы в окончательную матрицу изображения. Ваша задача — переписать этот код так, чтобы он работал как можно быстрее, используя техники вынесения части кода программы, разворачивания циклов и разбиения на блоки (рекомендуется к самостоятельному изучению).

Вы можете найти этот код в файле `kernels.c`.

Сглаживание

Функция “сглаживания” берёт исходное изображение `src` и возвращает “сглаженный” результат в окончательном изображении `dst`. Ниже предоставлена часть этой реализации:

```
void naive_smooth(int dim, pixel *src, pixel *dst) {
    int i, j;

    for(i=0; i < dim; i++)
        for(j=0; j < dim; j++)
            dst[RIDX(i,j,dim)] = avg(dim, i, j, src); /* Smooth the (i,j)th pixel */
return;
}
```

Функция `avg` возвращает среднее значение пикселей вокруг пикселя `(i,j)`. Ваша задача — оптимизировать `smooth` (и `avg`), чтобы они работали как можно быстрее. (*Примечание*: `avg` — это локальная функция, и вы можете полностью избавиться от нее, чтобы реализовать `smooth` каким-либо другим образом.)

Этот код (и реализация `avg`) находятся в файле `kernels.c`.

Измерение производительности

Нашей главной метрикой производительности является *CPE* (*Cycles per Element — Циклы на элемент*). Значение *CPE* вычисляется по следующей формуле: *C/N^2*, где *C* — это количество циклов, взятое функцией, а *N^2* — размер изображения (*N x N*). Таблица 1 приводит производительность наивных (прямых) способов реализации, указанных выше, и сравнивает их с оптимизированным способом реализации. Производительность показана на 5 различных значениях *N*. Все вычисления проводились на машине “Pentium III Xeon Fish”.

Соотношение (ускорение) оптимизированного и наивного способов реализации показывает успешность вашей реализации. Чтобы суммировать общий эффект на различных значениях *N*, мы вычислим среднее геометрическое результатов этих пяти значений. То есть, если измеренные ускорения для *N* = {32, 64, 128, 256, 512} равняются R(32), *R(64)*, *R(128)*, *R(256)* и *R(5126)*, то мы вычисляем общую производительность следующим способом:

$$R = \sqrt[5]{R_{32} \times R_{64} \times R_{128} \times R_{256} \times R_{512}}$$

Figure 1: вычисление среднего ускорения

Тест	1	2	3	4	5	
Значение N	64	128	256	512	1024	Среднее геом.
Наивная (прямая) `rotate` (CPE)	14.7	40.1	46.4	65.9	94.5	
Оптимизированная `rotate` (CPE)	8.0	8.6	14.8	22.1	25.3	
Ускорение (наивная/оптимизированная)	1.8	4.7	3.1	3.0	3.7	3.1
Значение N	32	64	128	256	512	Среднее геом.
Наивная (прямая) `smooth` (CPE)	695	698	702	717	722	
Оптимизированная `smooth` (CPE)	41.5	41.6	41.2	53.5	56.4	
Ускорение (наивная/оптимизированная)	16.8	16.8	17.0	13.4	12.8	15.2

Таблица 1: CPE и соотношения оптимизированных и наивных (прямых) способов реализации

Предположения

Для упрощения, вы можете предположить, что N кратно 32. Ваш код должен работать без ошибок для всех подобных значений N, но мы измерим его производительность только для пяти значений, указанных в таблице 1.

4. Инфраструктура

Мы предоставили код, который поможет вам протестировать корректность ваших реализаций, а также измерить их производительность. Этот раздел описывает, как использовать эту инфраструктуру. Точные детали каждой части задания описаны в следующем разделе.

Примечание: Единственные два исходных файла, который вы будете модифицировать — это `kernels.c` и `config.h`

Управление версиями

Вы напишете множество версий процедур `rotate` и `smooth`. Чтобы помочь вам сравнить производительность всех различных версий, которые вы написали, мы предоставляем вам способ "регистрации" функций.

К примеру, файл `kernels.c` содержит следующую функцию:

```
void register_rotate_functions() {
    add_rotate_function(&rotate, rotate_descr);
}
```

Эта функция содержит одно или несколько обращений к `add_rotate_function`. В примере выше, `add_rotate_function` регистрирует функцию `rotate` вместе со строкой `rotate_descr`, которая является ASCII-описанием того, что

делает функция. Просмотрите файл `kernels.c`, чтобы увидеть, как создавать описания строк. Эта строка может содержать в себе не более, чем 256 символов.

Аналогичная функция сглаживания содержится в файле `kernels.c`.

Драйвер

Исходный код, который вы напишите, будет связан с объектным кодом, который передается в двоичную систему `driver`. Чтобы создать эту двоичную систему, вы должны выполнить следующую команду:

```
unix> make driver
```

Вы должны пересоздавать драйвер каждый раз, когда вы меняете код в `kernels.c`. Чтобы проверить ваши реализации, вы можете запустить команду:

```
unix> ./driver
```

5. Детали задания

Оптимизация поворота

В этой части, вы будете оптимизировать `rotate`, чтобы достичь как можно более низкого показателя *CPE*. Вы должны скомпилировать `driver`, а затем запустить его с соответствующими аргументами, чтобы протестировать ваши реализации.

К примеру, если запустить драйвер с предоставленной наивной версией (функции `rotate`), то он генерирует следующий вывод:

```
unix> ./driver

Rotate: Version = naive_rotate: Naive baseline implementation:
Dim             64      128     256     512     1024    Mean
Your CPEs       14.6    40.9    46.8    63.5    90.9
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         1.0     1.0     1.0     1.0     1.0     1.0
```

Важно! Замените значения макроопределний в строках 15-19 файла config.h значениями из строки `Your CPEs` для функции naive_rotate. Эти значения будут использоваться для вычисления ускорения ваших реализаций.

Оптимизация сглаживания

В этой части, вы будете оптимизировать `smooth`, чтобы достичь как можно более низкого показателя *CPE*.

К примеру, если запустить драйвер с предоставленной наивной версией (функции `smooth`), то он генерирует следующий вывод:

```
unix> ./driver

Rotate: Version = naive_smooth: Naive baseline implementation:
Dim             32      64      128     256     512     Mean
Your CPEs       695.8   698.5   703.8   720.3   722.7
Baseline CPEs   695.0   698.0   702.0   717.0   722.0
Speedup         1.0     1.0     1.0     1.0     1.0     1.0
```

Важно! Замените значения макроопределний в строках 27-31 файла config.h значениями из строки `Your CPEs` для функции naive_rotate. Эти значения будут использоваться для вычисления ускорения ваших реализаций.

Совет. Посмотрите на ассемблерный код, сгенерированный на функциях `rotate` и `smooth` (прим. `godbolt.org`). Сосредоточьтесь на оптимизации внутреннего цикла (кода, который повторно выполняется в цикле), используя техники оптимизации, пройденные на уроке. Функция `smooth` больше полагается на вычисления, и меньше полагается на память по сравнению с функцией `rotate`, поэтому способы оптимизации будут выглядеть по-разному.

Правила написания кода

Вы можете написать любой по вашему желанию код в рамках следующих правил:

- Он должен быть написан на ANSI Си. Вы не можете использовать встроенные операторы ассемблерных языков.
- Он не должен мешать механизму измерения. Не допускается, чтобы код выводил любую постороннюю информацию.

Вы можете модифицировать код только в файле `kernels.c`. Вам разрешено определять макросы, глобальные переменные и другие процедуры в этих файлах.

Оценка

Ваши решения для `rotate` и `smooth` составляют по 50% от вашей оценки. Балл для каждого из решений будет основан на следующих критериях:

- Правильность: Вы НЕ ПОЛУЧИТЕ БАЛЛЫ за код, имеющий в себе ошибки, которые приводят к тому, что драйвер не функционирует как должен. Это также включает в себя код, который корректно работает

с тестовыми размерами, но неправильно с матрицами изображений других размеров. Как упомянуто ранее, вы можете предположить, что измерение изображения кратно 32.

- *CPE*: Вы получите полное количество баллов за ваши реализации функций `rotate` и `smooth`, если они корректны и достигают средних значений *CPE* выше пороговых значений `S(r) = 1.5` и `S(s) = 3`. Вы получите баллы частично за корректную реализацию, которая работает лучше, чем изначательно предоставленная.