CSAPP

实 验 报 告

学生姓名　　吴语港

学生学号　　SA19225404

实验日期　　2019/10/29

实　验　报　告

一、实验名称：perflab

二、实验学时： 3

三、实验内容和目的：

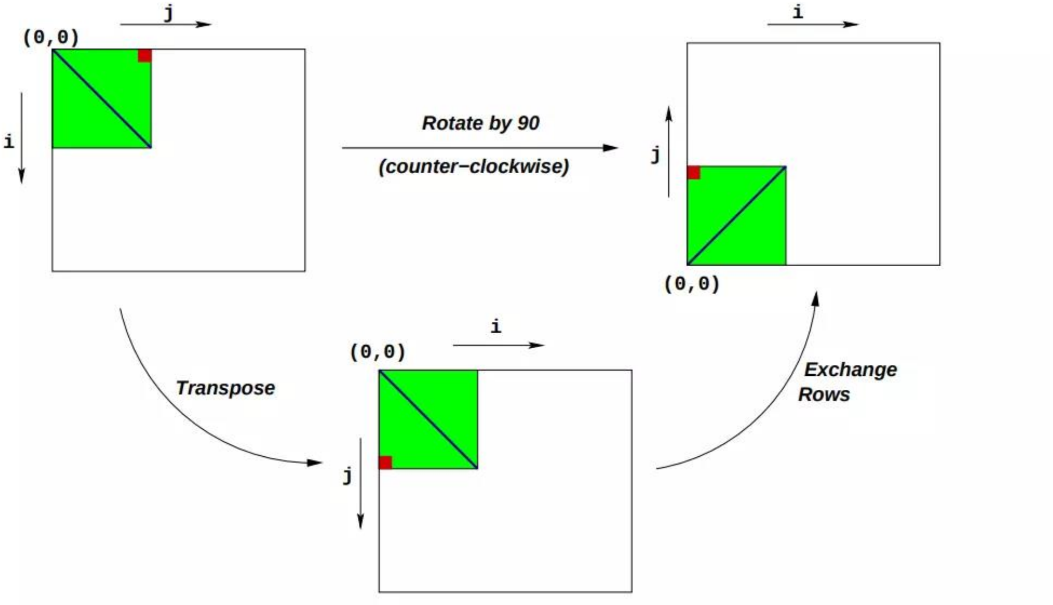
此次实验进行图像处理代码的优化。图像处理提供了许多能够通过优化而得到改良的函数。在此次实验中，我们将考虑两种图像处理操作：**roate**，　此函数用于将图像逆时针旋转 90°；以及 **smooth**，对图像进行“平滑”或者说“模糊”处理。

对于此次实验，我们将认为图像是以一个二维矩阵 M 来表示的，并以 $M_{i,j}$ 来表记（i，j）位置的像素值。像素值是由红，绿，蓝（RGB）三个值构成的三元组。我们仅考虑方形图像。以 N 来表记图像的行数（同时也是列数）。行和列均以 C 风格进行编号——从 0 到 $N-1$ 。

在这种表示方法之下，**rotate** 操作可以借由以下两种矩阵操作的结合来简单实现：

- 转置：对于每个（i，j），交换 $M_{i,j}$ 与 $M_{j,i}$ 。
- 行交换：交换第 i 行与第 $N-1-i$ 行。

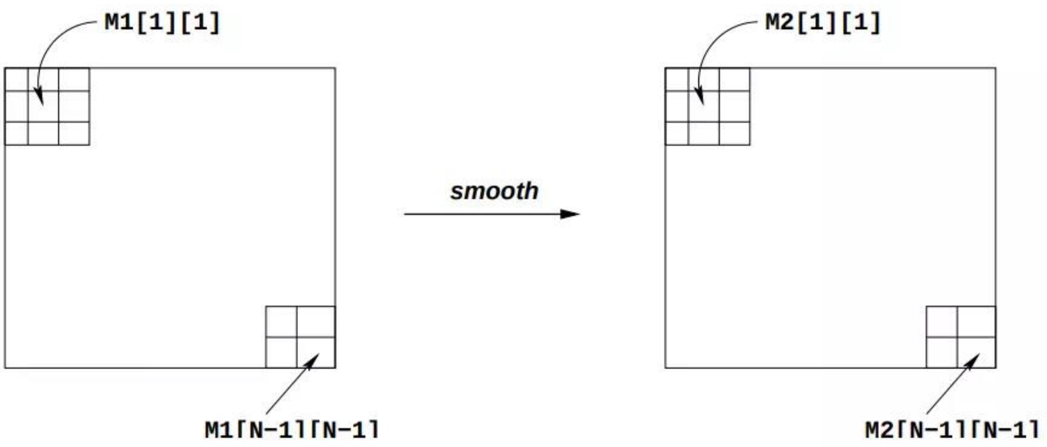
详情见下图：



smooth 操作可以通过求每个像素与周围像素（最多是以该像素为中心的 3×3 的九宫格）的均值。详见图 2，像素 M2[1][1] 与 M2[N - 1][N - 1] 由下式给出：

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

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



四、实验步骤及结果：

首先将 perflab-handout.tar 拷贝至一个受保护的文件夹，用于完成此次实验。

然后在命令行输入命令：tar xvf perflab-handout.tar 这将使数个文件被解压至当前目录。

你可以进行修改并最终提交的唯一文件是 kernels.c 程序 driver.c 是一个驱动程序，你可以用它来评估你解决方案的性能。

使用命令：make driver 来生成驱动代码，并用命令 ./driver 来使其运行。

查看文件 kernels.c，你会发现一个 C 结构体：team。你需要将你小组成员的信息填入其中。请马上填写，以防你忘记。

1.naive_rotate

分析原代码：

```
/*
 *naive_rotate - The naive baseline version of rotate
 */
char naive_rotate_descr[] ="naive_rotate: Naive baseline
implementation";
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)];
}
```

在头文件 defs.h 中找到了宏定义：

```
#defineRIDX(i,j,n) ((i)*(n)+(j))
```

那么这段代码就很容易为一幅画的旋转，它将将所有的像素进行行列调位、导致整幅图画进行了 90 度旋转。

但是由于这串代码的步长太长，导致 cache 的命中率非常低，所以总体运算效率低。因此，我们考虑到 cache 的大小，应在存储的时候进行 32 个像素依次存储（列存储）。（32 个像素排列是为了充分利用一级缓存(32KB)，采用分块策略，每一个块大小为 32）

这样可以做到 cache 友好、可以大幅度提高效率。

2.perf_rotate

考虑矩形分块 32*1，32 路循环展开，并使 dest 地址连续，以减少存储器写次数

```
//宏定义一个复制函数，方便程序编写
#define COPY(d,s) *(d)=*(s)
```

```
char rotate_descr[] = ″rotate: Currentworking version″;

void rotate( int dim,pixel *src,pixel *dst)
{
   int i,j;
   for(i=0;i<dim;i+=32)//32路循环展开，32个像素依次存储
         for(j=dim-1;j>=0;j-=1)
{
         pixel*dptr=dst+RIDX(dim-1-j,i,dim);
         pixel*sptr=src+RIDX(i,j,dim);
//进行复制操作
         COPY(dptr,sptr);sptr+=dim;
         COPY(dptr+1,sptr);sptr+=dim;
         COPY(dptr+2,sptr);sptr+=dim;
         COPY(dptr+3,sptr);sptr+=dim;
         COPY(dptr+4,sptr);sptr+=dim;
         COPY(dptr+5,sptr);sptr+=dim;
         COPY(dptr+6,sptr);sptr+=dim;
         COPY(dptr+7,sptr);sptr+=dim;
         COPY(dptr+8,sptr);sptr+=dim;
         COPY(dptr+9,sptr);sptr+=dim;
         COPY(dptr+10,sptr);sptr+=dim;
         COPY(dptr+11,sptr);sptr+=dim;
         COPY(dptr+12,sptr);sptr+=dim;
         COPY(dptr+13,sptr);sptr+=dim;
         COPY(dptr+14,sptr);sptr+=dim;
         COPY(dptr+15,sptr);sptr+=dim;
         COPY(dptr+16,sptr);sptr+=dim;
         COPY(dptr+17,sptr);sptr+=dim;
         COPY(dptr+18,sptr);sptr+=dim;
         COPY(dptr+19,sptr);sptr+=dim;
         COPY(dptr+20,sptr);sptr+=dim;
         COPY(dptr+21,sptr);sptr+=dim;
         COPY(dptr+22,sptr);sptr+=dim;
         COPY(dptr+23,sptr);sptr+=dim;
         COPY(dptr+24,sptr);sptr+=dim;
         COPY(dptr+25,sptr);sptr+=dim;
         COPY(dptr+26,sptr);sptr+=dim;
         COPY(dptr+27,sptr);sptr+=dim;
         COPY(dptr+28,sptr);sptr+=dim;
         COPY(dptr+29,sptr);sptr+=dim;
         COPY(dptr+30,sptr);sptr+=dim;
         COPY(dptr+31,sptr);
}
}
```

3.smooth

分析原代码

```
char naive_smooth_descr[] ="naive_smooth: Naive baseline
implementation";
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);
}
```

这段代码频繁地调用 avg 函数，并且 avg 函数中也频繁调用 initialize_pixel_sum 、accumulate_sum、assign_sum_to_pixel 这几个函数，且又含有 2 层 for 循环，而我们应该减少函数调用的时间开销。所以，需要改写代码，不调用 avg 函数。

特殊情况，特殊对待：

Smooth 函数处理分为 4 块，一为主体内部，由 9 点求平均值；二为 4 个顶点，由 4 点求平均值；三为四条边界，由 6 点求平均值。从图片的顶部开始处理，再上边界，顺序处理下来，其中在处理左边

界时，for 循环处理一行主体部分，于是就有以下优化的代码。

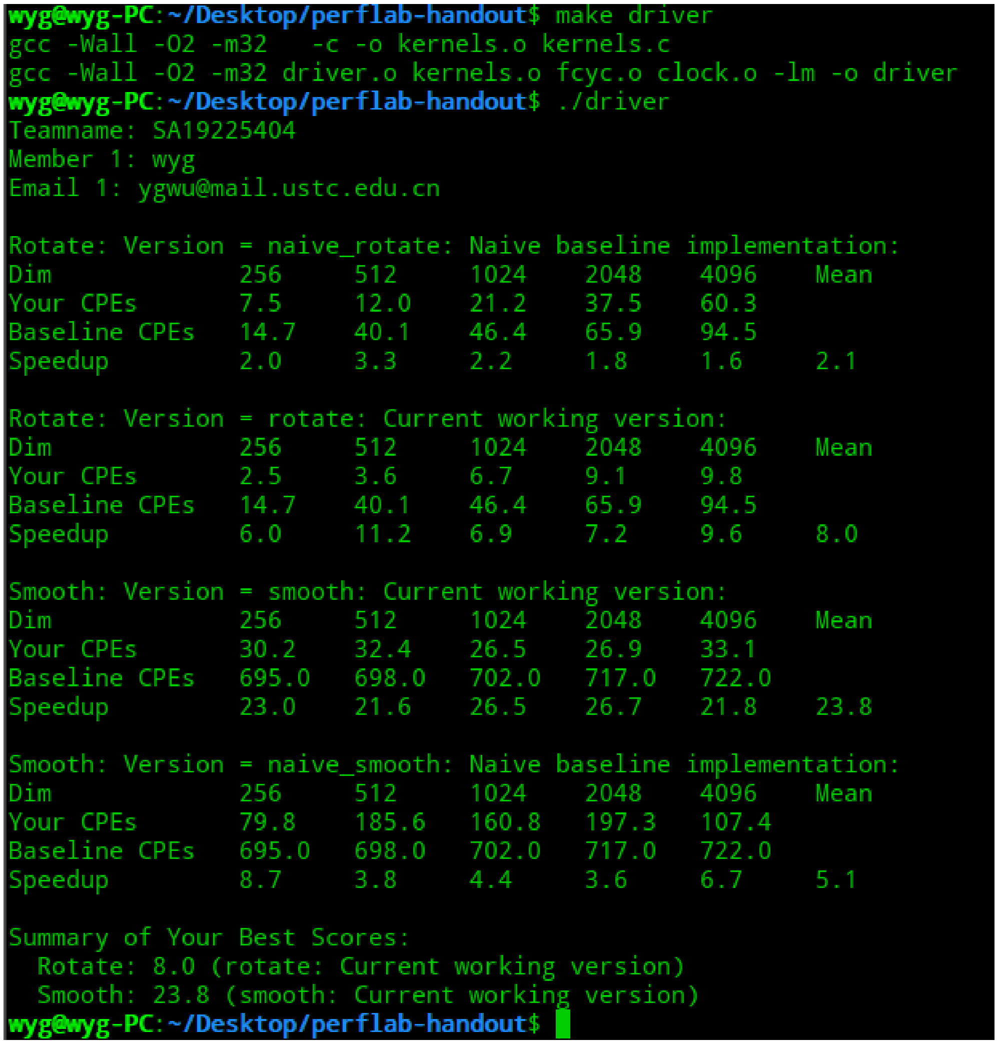
4.perf_smooth

```
charsmooth_descr[] = ″smooth: Current working version″;
void smooth(intdim, pixel *src, pixel *dst)  {
         int i,j;
         int dim0=dim;
         int dim1=dim-1;
         int dim2=dim-2;
         pixel *P1, *P2, *P3;
         pixel *dst1;
         P1=src;
         P2=P1+dim0;
//左上角像素处理        采用移位运算代替 avg 的某些操作
dst->red=(P1->red+(P1+1)->red+P2->red+(P2+1)->red)>>2;
dst->green=(P1->green+(P1+1)->green+P2->green+(P2+1)->gre
en))>>2;
dst->blue=(P1->blue+(P1+1)->blue+P2->blue+(P2+1)->blue)>>
2;
dst++;
//上边界处理
for(i=1;i<dim1;i++)  {
dst->red=(P1->red+(P1+1)->red+(P1+2)->red+P2->red+(P2+1)-
>red+(P2+2)->red)/6;
dst->green=(P1->green+(P1+1)->green+(P1+2)->green+P2->gre
en+(P2+1)->green+(P2+2)->green)/6;
dst->blue=(P1->blue+(P1+1)->blue+(P1+2)->blue+P2->blue+(P
2+1)->blue+(P2+2)->blue)/6;
                   dst++;
                   P1++;
                   P2++;
         }
//右上角像素处理
dst->red=(P1->red+(P1+1)->red+P2->red+(P2+1)->red)>>2;
```

```
dst->green=(P1->green+(P1+1)->green+P2->green+(P2+1)->green)>>2;
dst->blue=(P1->blue+(P1+1)->blue+P2->blue+(P2+1)->blue)>>2;
       dst++;
       P1=src;
       P2=P1+dim0;
       P3=P2+dim0;
//左边界处理
for(i=1;i<dim1;i++)  {
dst->red=(P1->red+(P1+1)->red+P2->red+(P2+1)->red+P3->red+(P3+1)->red)/6;
dst->green=(P1->green+(P1+1)->green+P2->green+(P2+1)->green+P3->green+(P3+1)->green)/6;
dst->blue=(P1->blue+(P1+1)->blue+P2->blue+(P2+1)->blue+P3->blue+(P3+1)->blue)/6;
                      dst++;
                      dst1=dst+1;
}
//主体中间部分处理
for(j=1;j<dim2;j+=2)     {
//同时处理2个像素

dst->red=(P1->red+(P1+1)->red+(P1+2)->red+P2->red+(P2+1)->red+(P2+2)->red+P3->red+(P3+1)->red+(P3+2)->red)/9;

dst->green=(P1->green+(P1+1)->green+(P1+2)->green+P2->green+(P2+1)->green+(P2+2)->green+P3->green+(P3+1)->green+(P3+2)->green)/9;

dst->blue=(P1->blue+(P1+1)->blue+(P1+2)->blue+P2->blue+(P2+1)->blue+(P2+2)->blue+P3->blue+(P3+1)->blue+(P3+2)->blue)/9;

dst1->red=((P1+3)->red+(P1+1)->red+(P1+2)->red+(P2+3)->re
d+(P2+1)->red+(P2+2)->red+(P3+3)->red+(P3+1)->red+(P3+2)-
>red)/9;

dst1->green=((P1+3)->green+(P1+1)->green+(P1+2)->green+(P
2+3)->green+(P2+1)->green+(P2+2)->green+(P3+3)->green+(P3
+1)->green+(P3+2)->green)/9;

dst1->blue=((P1+3)->blue+(P1+1)->blue+(P1+2)->blue+(P2+3)
->blue+(P2+1)->blue+(P2+2)->blue+(P3+3)->blue+(P3+1)->blu
e+(P3+2)->blue)/9;

dst+=2;dst1+=2;P1+=2;P2+=2;P3+=2;
                            }
                            for(;j<dim1;j++)      {

dst->red=(P1->red+(P1+1)->red+(P1+2)->red+P2->red+(P2+1)-
>red+(P2+2)->red+P3->red+(P3+1)->red+(P3+2)->red)/9;

dst->green=(P1->green+(P1+1)->green+(P1+2)->green+P2->gre
en+(P2+1)->green+(P2+2)->green+P3->green+(P3+1)->green+(P
3+2)->green)/9;

dst->blue=(P1->blue+(P1+1)->blue+(P1+2)->blue+P2->blue+(P
2+1)->blue+(P2+2)->blue+P3->blue+(P3+1)->blue+(P3+2)->blu
e)/9;
                                     dst++;
P1++;P2++;P3++;
                            }
//右侧边界处理
dst->red=(P1->red+(P1+1)->red+P2->red+(P2+1)->red+P3->red
+(P3+1)->red)/6;
dst->green=(P1->green+(P1+1)->green+P2->green+(P2+1)->gre
en+P3->green+(P3+1)->green)/6;
dst->blue=(P1->blue+(P1+1)->blue+P2->blue+(P2+1)->blue+P3
->blue+(P3+1)->blue)/6;
```

```
                            dst++;      P1+=2;      P2+=2;
P3+=2;
         }
//左下角处理
dst->red=(P1->red+(P1+1)->red+P2->red+(P2+1)->red)>>2;

dst->green=(P1->green+(P1+1)->green+P2->green+(P2+1)->gre
en)>>2;

dst->blue=(P1->blue+(P1+1)->blue+P2->blue+(P2+1)->blue)>>
2;
         dst++;
//下边界处理
for(i=1;i<dim1;i++)     {
dst->red=(P1->red+(P1+1)->red+(P1+2)->red+P2->red+(P2+1)-
>red+(P2+2)->red)/6;
dst->green=(P1->green+(P1+1)->green+(P1+2)->green+P2->gre
en+(P2+1)->green+(P2+2)->green)/6;
dst->blue=(P1->blue+(P1+1)->blue+(P1+2)->blue+P2->blue+(P
2+1)->blue+(P2+2)->blue)/6;
                   dst++;      P1++;     P2++;
         }
//右下角像素处理
dst->red=(P1->red+(P1+1)->red+P2->red+(P2+1)->red)>>2;
dst->green=(P1->green+(P1+1)->green+P2->green+(P2+1)->gre
en)>>2;
dst->blue=(P1->blue+(P1+1)->blue+P2->blue+(P2+1)->blue)>>
2;
//全部处理完毕
}
```

实验结果：



```
wyg@wyg-PC:~/Desktop/perflab-handout$ make driver
gcc -Wall -O2 -m32   -c -o kernels.o kernels.c
gcc -Wall -O2 -m32 driver.o kernels.o fcyc.o clock.o -lm -o driver
wyg@wyg-PC:~/Desktop/perflab-handout$ ./driver
Teamname: SA19225404
Member 1: wyg
Email 1: ygwu@mail.ustc.edu.cn

Rotate: Version = naive_rotate: Naive baseline implementation:
Dim             256     512     1024    2048    4096    Mean
Your CPEs       7.5     12.0    21.2    37.5    60.3
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         2.0     3.3     2.2     1.8     1.6     2.1

Rotate: Version = rotate: Current working version:
Dim             256     512     1024    2048    4096    Mean
Your CPEs       2.5     3.6     6.7     9.1     9.8
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         6.0     11.2    6.9     7.2     9.6     8.0

Smooth: Version = smooth: Current working version:
Dim             256     512     1024    2048    4096    Mean
Your CPEs       30.2    32.4    26.5    26.9    33.1
Baseline CPEs   695.0   698.0   702.0   717.0   722.0
Speedup         23.0    21.6    26.5    26.7    21.8    23.8

Smooth: Version = naive_smooth: Naive baseline implementation:
Dim             256     512     1024    2048    4096    Mean
Your CPEs       79.8    185.6   160.8   197.3   107.4
Baseline CPEs   695.0   698.0   702.0   717.0   722.0
Speedup         8.7     3.8     4.4     3.6     6.7     5.1

Summary of Your Best Scores:
  Rotate: 8.0 (rotate: Current working version)
  Smooth: 23.8 (smooth: Current working version)
wyg@wyg-PC:~/Desktop/perflab-handout$ 
```

对于 rotate 操作平均加速了 8 倍

对于 smooth 操作平均加速了 23.8 倍