## 一、实验名称：perflab

## 二、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。

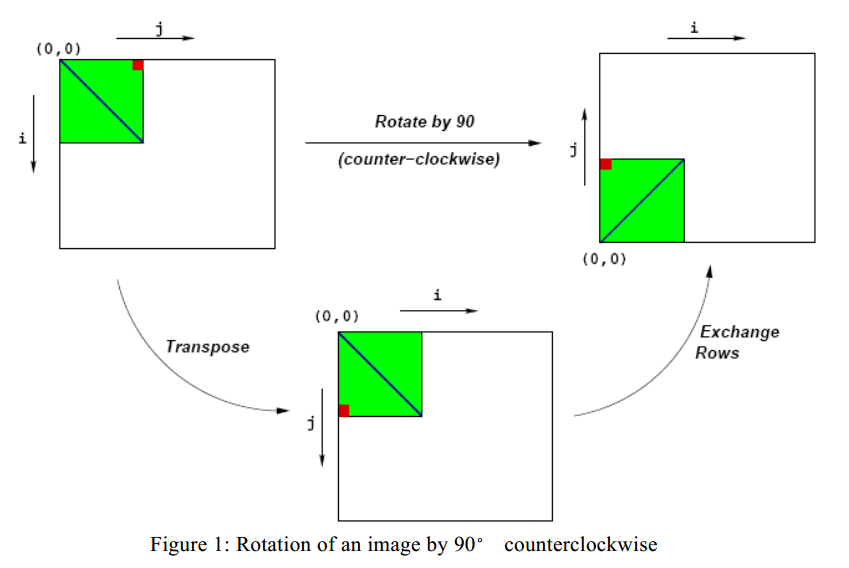
对于此次实验，我们将认为图像是以一个二维矩阵 M 来表示的，并以 Mi,j 来表记（i，j）位置的像素值。像素值是由红，绿，蓝（RGB）三个值构成的三元组。我们仅考虑方形图像。以 N 来表记图像的行数（同时也是列数）。行和列均以C风格进行编号——从0到 N - 1 。

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：

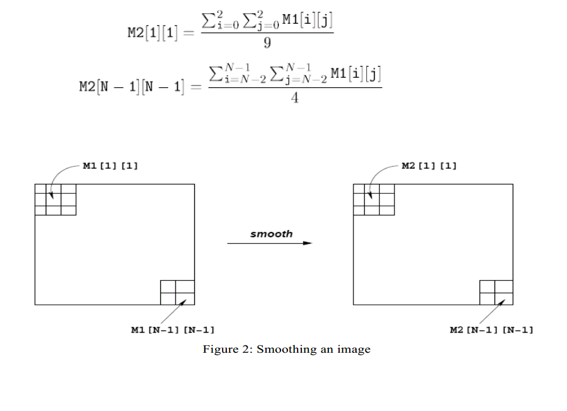
转置：对于每个（i，j），交换 Mi,j 与 Mj,i 。

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：



## 三、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，会发现一个C 结构体：team。需要将小组成员的信息填入其中。

## naive\_rotate

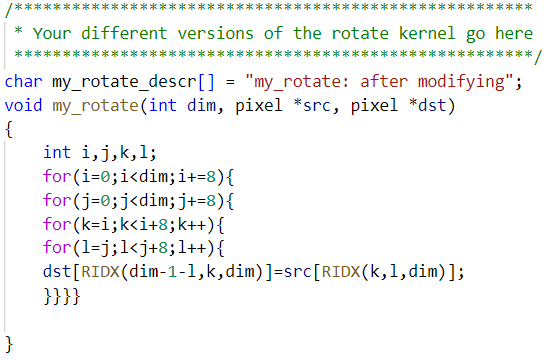
这段代码就很容易理解为一幅画的旋转，它将将所有的像素进行行列调位、导致整幅图画进行了90度旋转。

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）

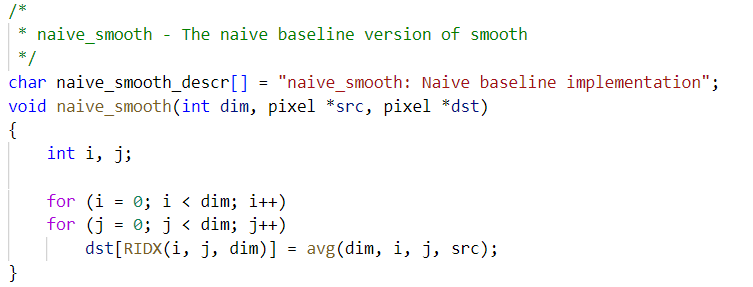
这样可以做到cache友好、可以大幅度提高效率。

## **my\_rotate**

考虑矩形分块8\*8分块，并使dest地址连续，以减少存储器写次数



## smooth



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。所以需要将函数调用拆分，不调用avg函数。

 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。

## my\_smooth



将循环调用的avg函数及其内部的assign\_sum\_to\_pixel等函数拆分，各部分在my\_smooth函数中依次处理减少函数调用，性能会得到提升，但代价是可读性会变得很低。

## 5. 实验结果

## 

对于rotate操作平均加速了24.3倍

 对于smooth操作平均加速了73.6倍