# 课程设计

## 目的

* 增进对计算机底层工作原理的理解
* 了解简单的性能优化技巧

## 环境

* VirtualBox
* Ubuntu

## 内容

### 背景介绍

本次实验进行内存敏感代码的优化。在本次实验中，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。

本次实验中，图像以一个二维矩阵 M 来表示的，并以 Mi,j 来标记（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，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：

第一步：转置：对于每个（i，j），交换 Mi,j 与 Mj,i

第二步：行交换：交换第 i 行与第 N - 1 - i 行

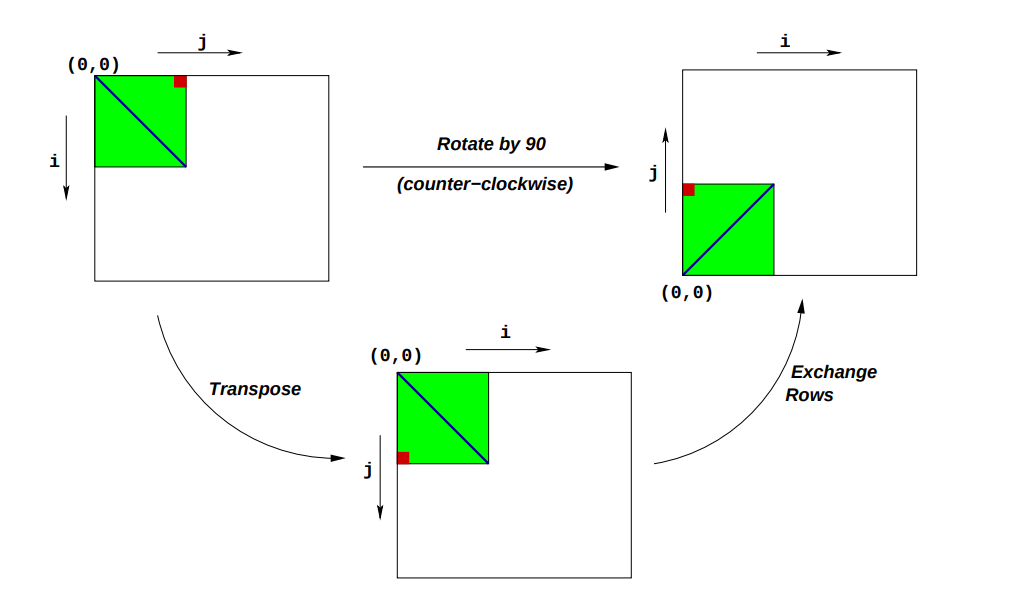
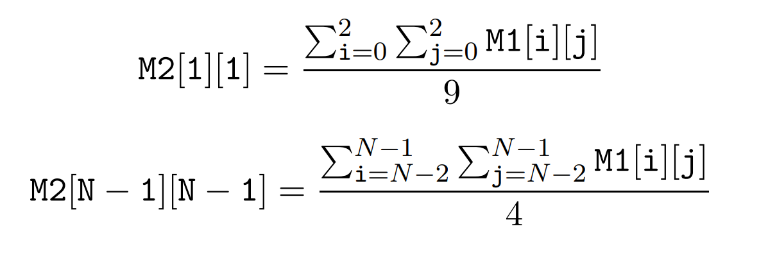


图1 将图像逆时针旋转90°

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）的均值来实现。详见图2，像素 M2[1][1] 与 M2[N - 1][N - 1] 由下式给出：



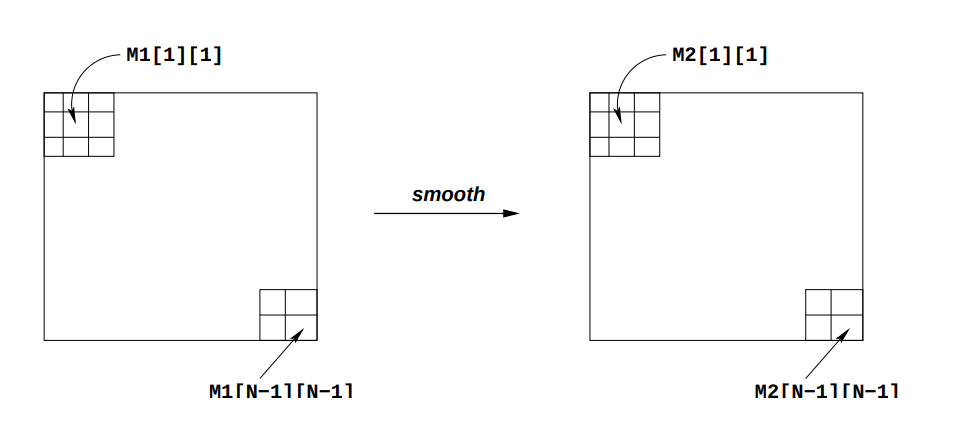
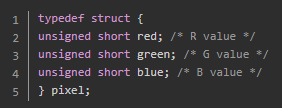


图2 平滑化图像

### 参考实现

图像表示的数据结构如下，像素被定义为以下结构体：

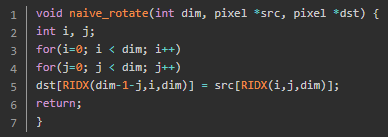


RGB 值拥有 16位的表示（16位色彩）。图像以一个一维的像素数组表示，（i，j）位置的像素表示为 I[RIDX(i,j,n)]。此处 n 表示图像矩阵的大小，RIDX 是一个宏，定义如下（像素的定义可查看defs.h的相关代码）：

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

**Rotate**

kernels.c的函数naive\_rotate用于计算源图像 **src** 旋转90°后的结果，并将结果保存在目标图像 **dst** 中。**dim** 表示图像的大小。

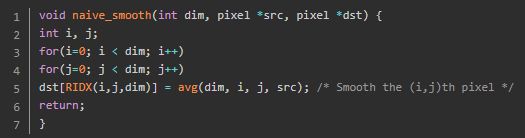


以上代码逐行扫描源图像，将元素拷贝至目标图像的列。

本函数是正确的，本次实验的任务是优化这段代码，通过 代码移动（code motion），循环展开（loop unrolling），分块技术（blocking）等技巧使其尽可能加速运行。

**smooth**

kernels.c的函数naive\_smooth传入源图像 **src** 作为参数，并以目标图像 **dst** 的形式返回平滑化的结果。此处是部分实现（其中**avg** 函数用于返回（i，j）位置周围像素的均值）：

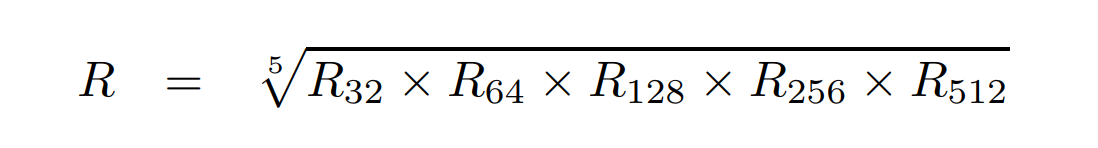


本函数是正确的，本次实验的任务是优化此函数（以及**avg**）函数，使其尽可能加速运行。（**avg** 是一个本地函数，你可以完全弃之不用，以其他方法实现**smooth**）

### 性能评测

性能评估的主要参数是 CPE，即每元素消耗的周期数（Cycles per Element）。如果函数在处理大小为 N×N 的图像时需要消耗C个周期，则 CPE 为 C/N2。表1总结了上面所写的原始方法的性能，并将其与优化过的实现方法进行了对比，一共有5组N值。以上的结果是在 Pentium III Xeon Fish 的机器上得到的。

基于优化实现与原始实现的比率（加速）产生不同版本代码的得分。为了总结不同N值的整体效果，最终会取每个结果的几何平均值。即：如果对于N={32, 64, 128, 256, 512}的加速分别为 R32，R64，R128， R256 以及 R512，则我们以下式计算总评得分：



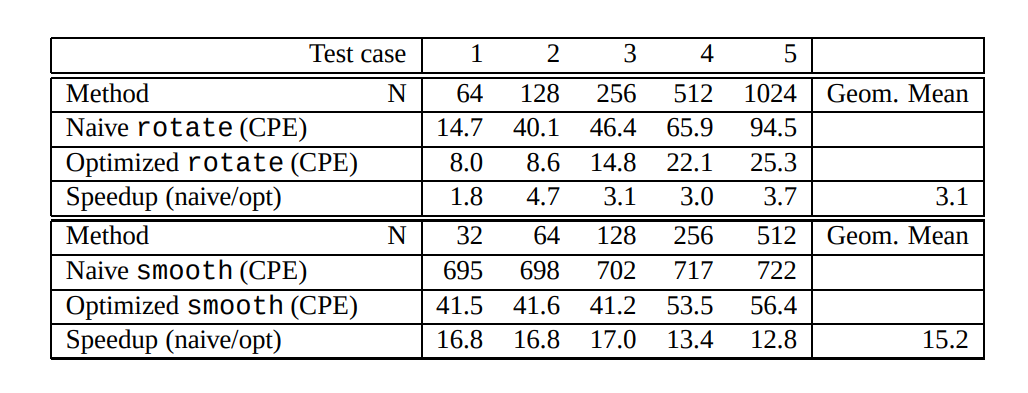


表1 优化版本vs.原始版本的CPE与比率

约定：为了简化问题，N总是32的倍数。要求优化后的代码必须对于这样的N值得到正确结果，但是性能评测只针对上表列出的5个值进行检测。

### 版本说明

kernels.c的函数naive\_rotate和naive\_smooth用来帮助检测你的优化代码是否结果正确，并衡量其性能。

注：允许改动的源文件只有 kernels.c。

为了帮助比较不同版本的优化代码的性能，允许写多个版本的**rotate** 及 **smooth**。每个函数需要通过register函数注册。

例如：kernels.c 中，提供了以下函数用于注册**rotate**的多个优化版本：



在此函数中，可以一次或多次调用 add\_rotate\_fuction 来注册不同版本的 **rotate** 函数，同时传入的参数还包括一个 ASCII 字符串，用于描述函数功能，此字符串最多允许键入256个字符。

各优化版本完成后，可以通过make生成名为 driver 的二进制文件。

driver 可以以多种模式运行：

* *默认模式*，运行函数的所有版本
* *自动评分模式*，此模式下只运行 **rotate( )** 与 **smooth( )** 函数。
* *文件模式*，此模式下，只有在输入文件中提及的函数会运行。
* *转存模式*，此模式下，会为每一个版本生成一行描述，并转存在一个文本文件中。然后可以编辑该文本文件，用于指定运行某些函数。

Driver的使用命令：如果不带任何参数，driver 将运行你全部的函数版本（默认模式）。其他模式如下：

-g：只运行 rotate( ) 与 smooth( ) 函数。（自动评分模式）。

-f <函数文件>：只执行在 <函数文件> 中指定的版本（文件模式）。

-d <转存文件>：将所有版本的名称转存到 <转存文件> 中，每个版本占一行（转存模式）。

-q：转存之后退出。与 -d 串联使用。例如，想在输出转存文件后立即退出，键入：./driver -qd dumpfile.

-h：打印命令行使用方法。

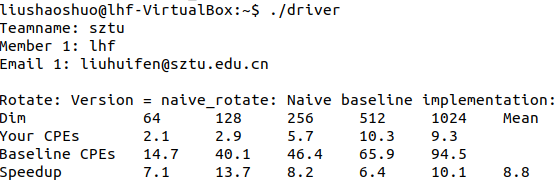
**重要：** 在开始之前，应当在 kernels.c 文件中的结构体填写信息（队名，成员，邮件地址）。

### 任务说明

1. 优化 rotate

要求优化 rotate 以使其达到尽可能低的CPE。

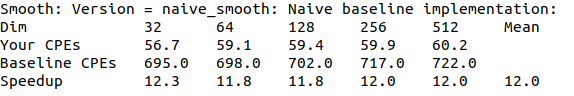
例如，以提供的 naive 版本运行 driver 会得到以下输出：

****

1. 优化smooth

要求优化 smooth，以使其达到尽可能低的CPE。

例如，以提供的naive 版本运行 driver 会得到以下输出：



**小建议** 查看rotate与smooth生成的对应汇编代码。重点关注对内部循环的优化（在循环内被反复执行的那些代码）。利用在课堂上学到的优化技巧。

比起 rotate， smooth是一个计算密集度更高而且内存敏感度更低的函数，所以优化方式会有所不同。

编码规则

你可以编写任何符合以下规则的代码：

* 必须使用 ANSI C，不可以使用嵌入式的汇编语言语句
* 不允许干扰时间测量机制
* 不要输出任何额外的内容
* 只能够修改kernels.c 文件中的代码，允许定义宏，全局变量以及其他过程

## 步骤

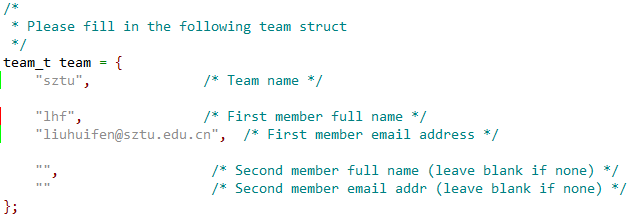
### 环境准备

新建自己的用户，并将本次实验的压缩包上传到环境中。

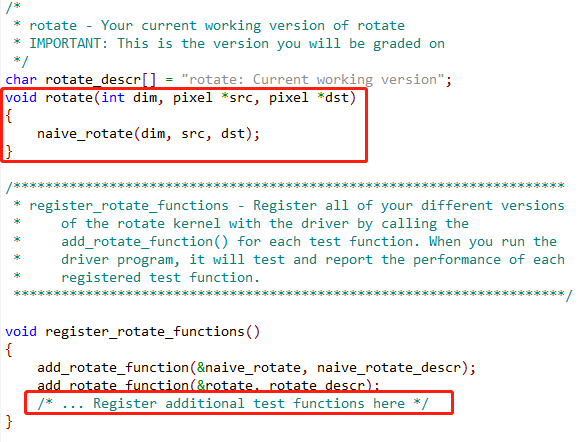
因为本次文件会解压到当前目录，故建议先新建一个目录，再将压缩包上传到新建的 目录下，最后再解压。

### 过程

1. 在kernels.c中编辑个人信息



1. 在kernels.c中通过register函数注册新增的优化函数，或者直接编辑rotate/smooth函数。



1. 生成driver，并运行。

make driver

unix> ./driver

注：每次修改kernels.c 文件之后，都需要重新生成 driver 文件。

1. 根据driver的运行结果，分析CPE和性能优化效果。

## 交付件要求

* 要求每个函数至少提供3个优化版本
* 提供每个版本的代码实现前的优化分析思路，以及运行之后的CPE数据分析
* 请将driver运行的结果截图
* 提供每个版本的最终代码，以文件的方式附在课程设计报告中
* 收集至少2个实际写代码中遇到的性能优化案例，并以此描述自己对于性能优化的理解