

**课程实验报告**

**题目： 实验报告**

**课程名称：**

**专业班级：**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期：**

**计算机科学与技术学院**

# 1实验一。。。。。

## 1.1实验目的与要求

（1）熟悉并行开发环境；

（2）了解并行编程的原则和方法；

（3）了解Linux下性能优化方法，如pthread、OpenMP、MPI等；

（4）掌握OpenCV处理图像的方法。

## 1.2实验内容

利用OpenCV编写一段图片处理程序，对图片进行锐化操作，不进行并行优化，记录处理时间，以便和后面实验的结果进行比较。

## 1.3实验方案

本次是对图像进行锐化操作，使用的卷积核为[-1, -1, -1, -1, 9, -1, -1, -1, -1]，而锐化操作其实就是遍历图像的每一个像素，计算它的邻域像素和卷积核矩阵的对应元素的乘积，然后加起来，作为该像素位置的值，便得到了新的图像。

遍历时要同时访问三行的像素数据，所以同时使用三个指针，分别指向当前行、上一行和下一行。通过uchar指针来访问每一个像素点，因为是彩色图像，所以同一行每个像素点间的间隔不是简单的加一，而是要加上每个像素的字节数，通过elemsize()属性来获得。因此程序就是简单的两层循环，外循环控制行，内循环控制列，遍历完后再将边界值赋0值即可。为了避免溢出，我们在运算外加上saturate\_cast<uchar>()。并在处理前后记录时间，相减后得到程序的运行时间，由于图片容量不大，我们将同一图片的锐化操作机械重复了10次，以增加运行时间，但实际只做了一次锐化处理。

得到了如下代码：

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <iostream>

using namespace cv;

using namespace std;

const int ctimes = 10;

int conv[] = {-1, -1, -1, -1, 9, -1, -1, -1, -1};

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

{

Mat \*ReadMat = new Mat[ctimes];

Mat \*Result = new Mat[ctimes];

for (int k = 0; k < ctimes; k++)

{

ReadMat[k] = imread("../1.jpg");

ReadMat[k].convertTo(ReadMat[k], CV\_8U);

Result[k].create(ReadMat[k].size(),ReadMat[k].type());

}

int stride;

double t = (double)getTickCount();

for (int k = 0; k < ctimes; k++)

{

stride = ReadMat[k].elemSize();

for(int j = 1 ; j < ReadMat->rows-1; j++)

{

const uchar\* previous=ReadMat[k].ptr<uchar>(j-1);// previous row

const uchar\* current=ReadMat[k].ptr<uchar>(j);// current row

const uchar\* next=ReadMat[k].ptr<uchar>(j+1);// next row

uchar\* output = Result[k].ptr<uchar>(j);// output row

for(int i= stride;i < stride\*(ReadMat[k].cols-1); i++)

{

\*output++ = saturate\_cast<uchar>(current[i] \* conv[4] + previous[i] \* conv[1] + next[i] \* conv[7]

+ current[i-stride] \* conv[3] + current[i+stride] \* conv[5]

+ previous[i-stride] \* conv[0] + previous[i+stride] \* conv[2]

+ next[i-stride] \* conv[6] + next[i+stride] \* conv[8]);

}

}

// Set the unprocess pixels to 0

Result[k].row(0).setTo(cv::Vec3b(0, 0, 0));

Result[k].row(Result[k].rows-1).setTo(cv::Vec3b(0, 0, 0));

Result[k].col(0).setTo(cv::Vec3b(0, 0, 0));

Result[k].col(Result[k].cols-1).setTo(cv::Vec3b(0, 0, 0));

}

t = ((double)getTickCount() - t)/getTickFrequency();

cout << "compute time: " << t << "s" << endl;

imwrite("1.jpg",Result[0]);

return 0;

}

通过xshell将程序上传、编译、运行测试时间即可。

## 1.4实验结果（介绍实验数据集，实验结果及分析）

本次试验的原始图片如图1-1所示。



图1-1 原始图片

经过锐化程序处理后得到的图片如图1-2所示。



图1-2 锐化后图片

对比可见锐化效果明显，图片处理成功。同时，程序的运算时间如图1-3所示。

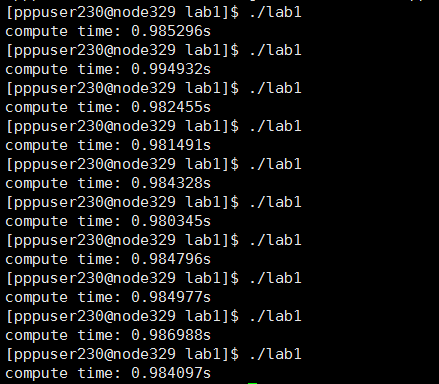


图1-3 实验1运算时间

如图所示，为了使实验结果更具一般性，我们将程序运行了十次，最终得到的平均运算时间为：

(0.985296+0.994932+0.982455+0.981491+0.984328+0.980345+0.984796+0.984977+0.986988+0.984097)/10=0.9849705(s)