Perflab3 实验日志

代码一（带详细注释）20%

```
260 char smooth_descr1[] = "smooth: Current working version 1.0";
261 void smooth1(int dim, pixel *src, pixel *dst)
262 {
263     int i, j, ii, jj, max_1, max_2, min_1, min_2;
264     pixel_sum sum;
265     pixel current_pixel;
266
267     for (i = 0; i < dim; i++)
268         for (j = 0; j < dim; j++)
269         {
270             sum.red = sum.green = sum.blue = 0, sum.num = 0;
271             max_1 = max(i-1, 0); max_2 = max(j-1,0);
272             min_1 = min(i+1,dim-1);min_2 = min(j+1, dim-1);
273             for(ii = max_1; ii <= min_1; ii++)
274                 for(jj = max_2; jj <= min_2; jj++)
275                 {
276                     sum.red += (int) src[RIDX(ii, jj, dim)].red;
277                     sum.green += (int) src[RIDX(ii, jj, dim)].green;
278                     sum.blue += (int) src[RIDX(ii, jj, dim)].blue;
279                     sum.num++;
280                 }
281             current_pixel.red = (unsigned short) (sum.red/sum.num);
282             current_pixel.green = (unsigned short) (sum.green/sum.num);
283             current_pixel.blue = (unsigned short) (sum.blue/sum.num);
284             dst[RIDX(i, j, dim)] = current_pixel;
285         }
286 }
```

详细注释：

（1） 将初始化函数用 avg 直接实现 减少函数调用的最大最小值函数

（2） 将计数函数也直接实现

（3） 提前计算 max 和 min 函数的结果

优化思路：

（1） 减少函数调用

（2） 提前计算函数的结果

实现过程：

（1） 比之前的优化多加平均值函数和技术函数，都可以直接实现

（2） 将 max 和 min 函数提前计算

运行速度：

```
Smooth: Version = smooth: Current working version 1.0:
Dim             32      64      128     256     512     Mean
Your CPEs       50.3    49.8    49.7    48.8    50.1
Baseline CPEs   695.0   698.0   702.0   717.0   722.0
Speedup         13.8    14.0    14.1    14.7    14.4    14.2
```

代码二（带详细注释）20%

```
char smooth_descr2[] = "smooth: Current working version 2.0";
void smooth2(int dim, pixel *src, pixel *dst)
{
    int i, j, ii, jj;
    pixel_sum sum;
    pixel current_pixel;
    for (i = 0; i < dim; i++)
        for (j = 0; j < dim; j++)
        {
            sum.red = sum.green = sum.blue = 0, sum.num = 0;
            if(i==0)
            {
                if(j==0)
                {
                    sum.red = sum.red + (int)src[RIDX(0,0,dim)].red +
                              (int)src[RIDX(0,1,dim)].red +
                              (int)src[RIDX(1,0,dim)].red +
                              (int)src[RIDX(1,1,dim)].red;
                    sum.green = sum.green + (int)src[RIDX(0,0,dim)].green +
                              (int)src[RIDX(0,1,dim)].green +
                              (int)src[RIDX(1,0,dim)].green +
                              (int)src[RIDX(1,1,dim)].green;
                    sum.blue = sum.blue + (int)src[RIDX(0,0,dim)].blue +
                              (int)src[RIDX(0,1,dim)].blue +
                              (int)src[RIDX(1,0,dim)].blue +
                              (int)src[RIDX(1,1,dim)].blue;
                    sum.num = sum.num + 4;
                }
                else if(j==dim-1)
                {
                    sum.red = sum.red + (int)src[RIDX(0,dim-1,dim)].red +
                              (int)src[RIDX(0,dim-2,dim)].red +
                              (int)src[RIDX(1,dim-1,dim)].red +
                              (int)src[RIDX(1,dim-2,dim)].red;
                    sum.green = sum.green + (int)src[RIDX(0,dim-1,dim)].green +
                              (int)src[RIDX(0,dim-2,dim)].green +
                              (int)src[RIDX(1,dim-1,dim)].green +
                              (int)src[RIDX(1,dim-2,dim)].green;
                    sum.blue = sum.blue + (int)src[RIDX(0,dim-1,dim)].blue +
                              (int)src[RIDX(0,dim-2,dim)].blue +
                              (int)src[RIDX(1,dim-1,dim)].blue +
                              (int)src[RIDX(1,dim-2,dim)].blue;
                    sum.num = sum.num + 4;
                }
                else
                {
                    for(ii = 0; ii <= 1;ii++)
                        for(jj = j - 1; jj <= j + 1;jj++)
                        {
                            accumulate_sum(&sum, src[RIDX(ii, jj, dim)]);
                        }
                }
            }
            else if(i == dim - 1)
            {
                if(j == 0)
                {
                    sum.red = sum.red + (int)src[RIDX(i,j,dim)].red +
                              (int)src[RIDX(i,j+1,dim)].red +
                              (int)src[RIDX(i-1,j,dim)].red +
                              (int)src[RIDX(i-1,j+1,dim)].red;
                    sum.green = sum.green + (int)src[RIDX(i,j,dim)].green +
                              (int)src[RIDX(i,j+1,dim)].green +
                              (int)src[RIDX(i-1,j,dim)].green +
                              (int)src[RIDX(i-1,j+1,dim)].green;
                    sum.blue = sum.blue + (int)src[RIDX(i,j,dim)].blue +
                              (int)src[RIDX(i,j+1,dim)].blue +
                              (int)src[RIDX(i-1,j,dim)].blue +
                              (int)src[RIDX(i-1,j+1,dim)].blue;
                    sum.num = sum.num + 4;
                }
                else if(j == dim-1)
                {
                    sum.red = sum.red + (int)src[RIDX(i,j,dim)].red +
                              (int)src[RIDX(i,j-1,dim)].red +
                              (int)src[RIDX(i-1,j,dim)].red +
                              (int)src[RIDX(i-1,j-1,dim)].red;
                    sum.green = sum.green + (int)src[RIDX(i,j,dim)].green +
                              (int)src[RIDX(i,j-1,dim)].green +
                              (int)src[RIDX(i-1,j,dim)].green +
                              (int)src[RIDX(i-1,j-1,dim)].green;
                    sum.blue = sum.blue + (int)src[RIDX(i,j,dim)].blue +
                              (int)src[RIDX(i,j-1,dim)].blue +
                              (int)src[RIDX(i-1,j,dim)].blue +
                              (int)src[RIDX(i-1,j-1,dim)].blue;
                    sum.num = sum.num + 4;
                }
                else
                {
                    for(ii = dim - 2; ii <= dim - 1;ii++)
```

```
                        for(jj = j - 1; jj <= j + 1;jj++)
                        {
                            accumulate_sum(&sum, src[RIDX(ii, jj, dim)]);
                        }
                    }
                }
                else
                {
                    if(j == 0)
                        for(ii = i - 1; ii <= i + 1;ii++)
                            for(jj = 0; jj <= 1; jj++)
                            {
                                accumulate_sum(&sum, src[RIDX(ii, jj, dim)]);
                            }
                    else if(j == dim - 1)
                        for(ii = i - 1; ii <= i + 1;ii++)
                            for(jj = dim - 2; jj <= dim - 1; jj++)
                            {
                                accumulate_sum(&sum, src[RIDX(ii, jj, dim)]);
                            }
                    else
                        for(ii = i - 1; ii <= i + 1;ii++)
                            for(jj = j - 1; jj <= j + 1;jj++)
                            {
                                accumulate_sum(&sum, src[RIDX(ii, jj, dim)]);
                            }
                }
            current_pixel.red = (unsigned short) (sum.red/sum.num);
            current_pixel.green = (unsigned short) (sum.green/sum.num);
            current_pixel.blue = (unsigned short) (sum.blue/sum.num);
            dst[RIDX(i, j, dim)] = current_pixel;
        }
}
```

详细注释：

（1） 不求最大最小值 将所有情况统一放在循环中执行

（2） 通过判断语句来判断位置

优化思路：

（1） 优化程序结构

（2） 减少函数调用

（3） 循环展开每一步（左下 右下 下部边界 左部边界 右部边界 中间等等）

（4） 优化运算

实现过程：

（1） 优化程序结构 不求最大最小值 将所有情况统一放在循环中执行

（2） 减少函数调用 不用调用初始化 求平均值等函数 直接实现

（3） 循环展开 直接实现

运行速度：

```
Smooth: Version = smooth: Current working version 2.0:
Dim              32      64      128     256     512     Mean
Your CPEs        43.0    45.4    43.1    46.1    46.8
Baseline CPEs    695.0   698.0   702.0   717.0   722.0
Speedup          16.2    15.4    16.3    15.6    15.4    15.8
```

perflab 实验报告

一，比较所完成的 partA 三段优化程序的优缺点并详细说明（30%）

1，第一段优化

优点：

（1）针对代码的结构进行了优化，保持了代码的可读性

（2）速度优化不错，有提升

缺点：速度的优化提升有局限性，仍然有许多的提升空间

2，第二段优化：

优点：

（1）将输入数据进行了优化，保留了一定的代码可读性

（2）提升了运行速度

缺点：

（1）只能针对像素为 32 的倍数的特殊样例

（2）对于其他样例不能正确运行，有局限性

3，第三段优化

优点：将循环展开，减少循环次数 ，提升运行速度。

缺点：代码可读性不好，只能针对像素为 32 倍数的特殊样例，没有对其他情况的兼容，有局限性

二，比较所完成的 partB 三段优化程序的优缺点并详细说明（30%）

1，第一段优化：

优点：通过减少函数调用，变量提前计算 min,max 的值，小幅度优化了速度

缺点：用 max，min 的方式判断像素点的位置，代码冗长，可读性一般

同时速度提升有限 即没有优化可读性 没有有效优化效率

2，第二段优化：

优点：减少函数调用 提升运行速度

缺点：产生负优化现象 原因是题中调用函数的同时还使用了指针 ，

3，第三段优化：

优点：

（1）优化程序结构 ，将所有不同种类的位置分块实现

（2）将一些除法运算改为位运算，并在主函数中实现了所有函数

缺点：代码冗长，可读性不强

三，利用 Amdahl 定律分析 partA 和 partB 两段程序，找出对于两段程序影响最大的部分，并对此说明采用何种策略优化效果最佳。（40%）

Part A

partA 程序的目标：将像素按照特定要求（旋转 90 度）转移

影响最大的部分：转移算法

第一段优化：优化移动需转移像素的方式，没有优化到主体，因此优化程度不够。

第二段优化：将每次转移一个元素改为每次转移 32 个元素，提升效率较高。

第三段优化：将 32 个元素循环展开，没有优化到主体，提升的效率有限。
通过以上说明，我们可以知道，对于像素转移的优化是最为重要的。

Part B

partB 程序的目标：对每一个像素算平均值。
影响最大的部分：计算以及判断程序的位置（9 种不同的位置）
第一段优化：减少函数的调用，没有优化函数主体，程序提升十分有限。
第二段优化：只是简单提升了函数的判断方式，优化速度不快。
第三段优化：优化函数主体，取消判断。分块处理每一个程序的位置，提升了算法性能。计算减少像素重合的规模。
通过以上说明，对于像素的判断与像素求平均值的计算的优化是影响最大的优化。