perflab 实验日志

一、**代码一**（带详细注释）20%

```
char rotate_descr[] = "rotate: Current working version 0";
void rotate(int dim, pixel *src, pixel *dst)
{
    int i, j, temp;
    for (j = 0; j < dim; j++)   //将j循环放外边，i循环放里边，减少指针的移动次数
    {
        temp = dim-1-j;         //建立临时变量temp储存重复计算的dim-1-j
        for (i = 0; i < dim; i++)
            dst[RIDX(temp,i,dim)] = src[RIDX(i,j,dim)];
    }
}
```

如上图所示：d 进行了两部优化，第一步将循环 i 放在外边，循环 j 放在里边第二步提前计算需要重复计算的 dim-1-j，在循环里直接使用结果。

二、**代码二**（带详细注释）20%

```
char rotate_descr1[] = "rotate: Current working version 1.1";
void rotate1(int dim, pixel *src, pixel *dst)
{
    int i, j, ii, jj, temp;
    for(ii = 0; ii < dim; ii=ii+32)
    {
        for(jj = 0; jj < dim; jj=jj+32)
        {
            for(j = jj; j < jj+32;j++)
            {
                temp = dim-1-j;           //建立临时变量储存需要重复计算的数据
                for(i = ii; i < ii+32;i++)//每次循环32个数据，以求cache命中
                    dst[RIDX(temp,i,dim)]=src[RIDX(i,j,dim)];
            }
        }
    }
}
```

在代码 1 的基础上，通过 cache 的知识，一次进行 32 个，提升了 cache 命中

三、**代码三**

```
char rotate_descr2[] = "rotate: Current working version 2.0";
void rotate2(int dim, pixel *src, pixel *dst)
{
    int i, j;
    dst = dim*(dim-1)+dst;//设置转换地点的初始值
    for(i = 0; i < dim; i= i + 32)//每次变换32个数据，以求cache命中
    {//把32行作为一个划分界限，每次将这32行的第j列在一个循环内一起转换
        for(j = 0; j < dim; j++)//转换开始
        {
            *dst = *src; dst++; src+=dim;//一共这样操作32次
            *dst = *src; dst++; src+=dim;//每次转换完成后将转换源向下移动一格
            *dst = *src; dst++; src+=dim;//将接受转换的地点左移一位
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
```

```
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src; dst++; src+=dim;
            *dst = *src;
            src = src - dim*31 + 1;//转换源初始化，为下一次转换做准备
            dst = dst -dim -31;//转换目标点初始化，为下一次转换做准备
        }
        src = src + dim*31;//转换源向下移动32行
        dst = dst + dim*dim + 32;//转化目标点对于转换源进行对应
    }
}
```

在代码二的基础上，将循环展开，进一步提升效率

四、文字描述**代码一**优化程序的思路及实现过程 30%

- 将循环 i 放在外边，循环 j 放在里边

  该方法减少了每次指针的移动弄位置，当 j 为循环外围时，每次循环都要将指针移动一个横排那么长，当 i 为循环外围时，每次只需寻址位置+1 即可
- 减少重复计算

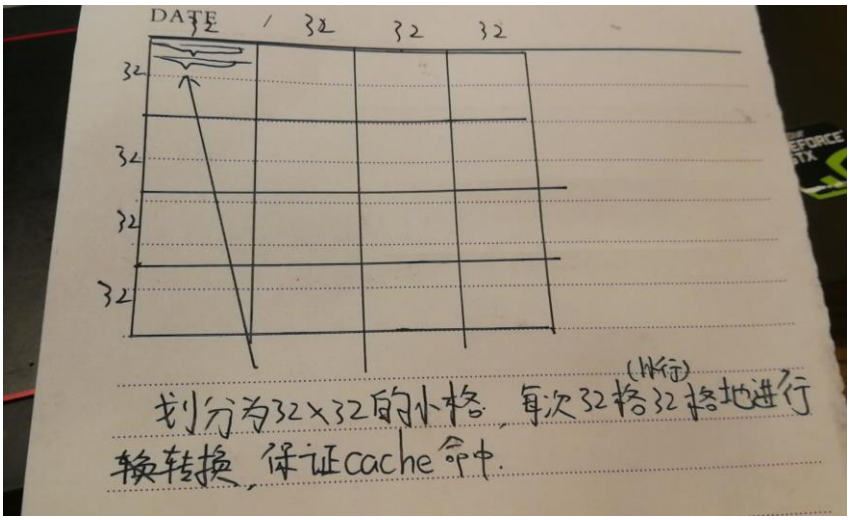
  我们注意到，在原本函数，中间有一步 dim-i-j 总是在重复运算，因此计划将其提前及计算，在循环中直接使用即可。

五、文字描述**代码二**优化程序的思路及实现过程 30%

- 优化思路

  根据 chahe 缓存的知识，将循环内部一次运行数目限制在 32 个，从而最大限度利用高速缓存中提升速度。
- 实现过程

  将图片划分为 32*32 像素的小块，每次转移一小行（32）个数据

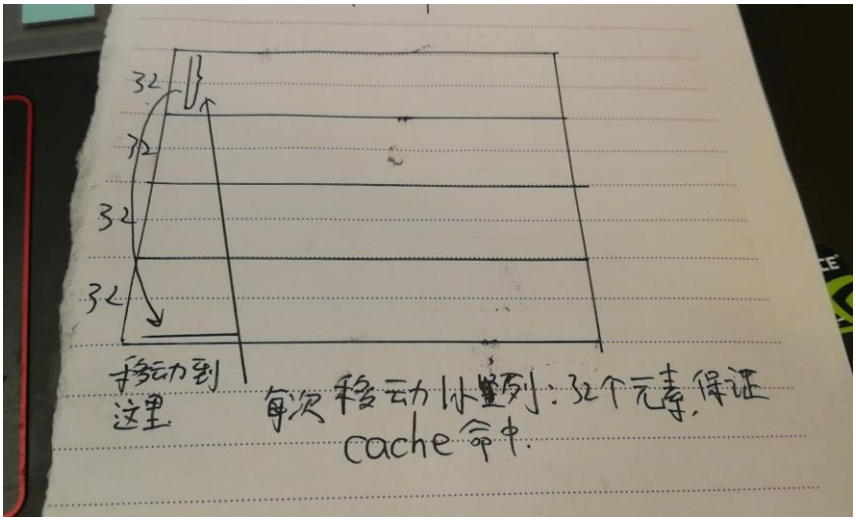


六、文字描述**代码三**优化程序的思路及实现过程

- 优化思路

  在代码二的基础上将循环展开，32 个移动直接敲出来，减少了程序中的跳转语句
- 实现过程

  将图片每 32 行作为一个划分，每次在 32 行中只移动 1 小列（32）个数据



七、运行速度提升截图

```
Rotate: Version = naive_rotate: Naive baseline implementation:
Dim             64      128     256     512     1024    Mean
Your CPEs       2.3     3.1     5.6     10.9    11.5
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         6.4     12.9    8.4     6.0     8.2     8.1

Rotate: Version = rotate: Current working version 0:
Dim             64      128     256     512     1024    Mean
Your CPEs       2.2     2.3     2.9     4.6     6.2
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         6.6     17.7    16.2    14.2    15.2    13.2

Rotate: Version = rotate: Current working version 1.1:
Dim             64      128     256     512     1024    Mean
Your CPEs       2.2     2.1     2.2     2.4     4.3
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         6.6     18.8    21.0    27.2    22.2    17.4

Rotate: Version = rotate: Current working version 2.0:
Dim             64      128     256     512     1024    Mean
Your CPEs       2.2     2.1     2.1     2.1     4.0
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         6.8     19.0    22.5    30.9    23.4    18.4
```