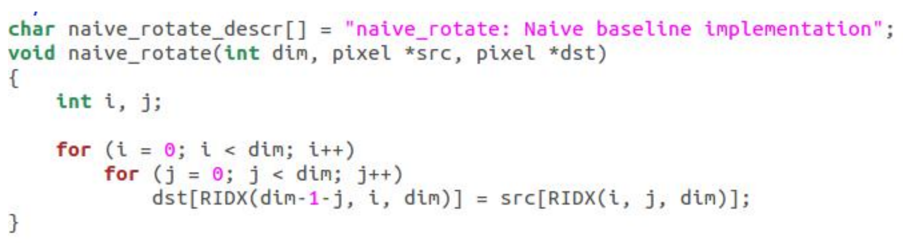
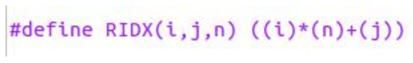
Perflab 实验日志一

一、源代码分析



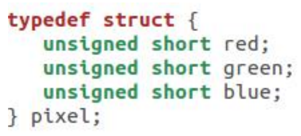
```c
char naive_rotate_descr[] = "naive_rotate: Naive baseline implementation";
void naive_rotate(int dim, pixel *src, pixel *dst)
{
    int i, j;

    for (i = 0; i < dim; i++)
        for (j = 0; j < dim; j++)
            dst[RIDX(dim-1-j, i, dim)] = src[RIDX(i, j, dim)];
}
```

rotate 函数是对一个大小为 dim*dim 的方块做逆时针旋转 90° 的操作。



```c
#define RIDX(i,j,n) ((i)*(n)+(j))
```

RIDX(i,j,n)即((i)*(n)+(j))。



```c
typedef struct {
   unsigned short red;
   unsigned short green;
   unsigned short blue;
} pixel;
```

pixel 是一个结构，src 为源图像，dst 为目标图像。

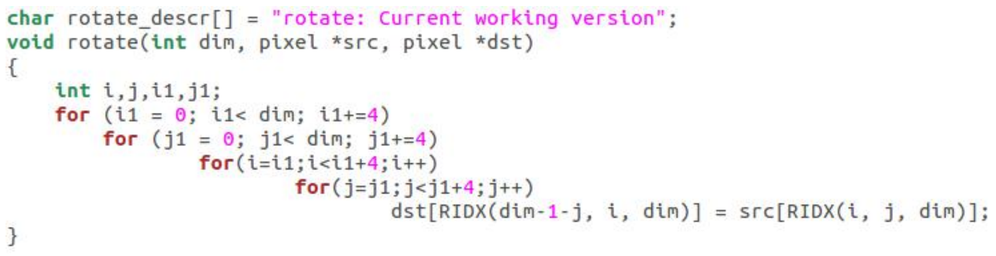
Rotate 函数的缺点为函数局部性不好，循环次数过多，可以对其循环部分进行分块或展开进行优化。

二、优化版本一

优化思路：循环分块

对循环分块，可以提高 cache 命中率，从而提高了空间局部性。从测试后的结果中也可看到，随着 dim 的增大，优化后的 CPE 比原代码的 CPE 增长得慢。

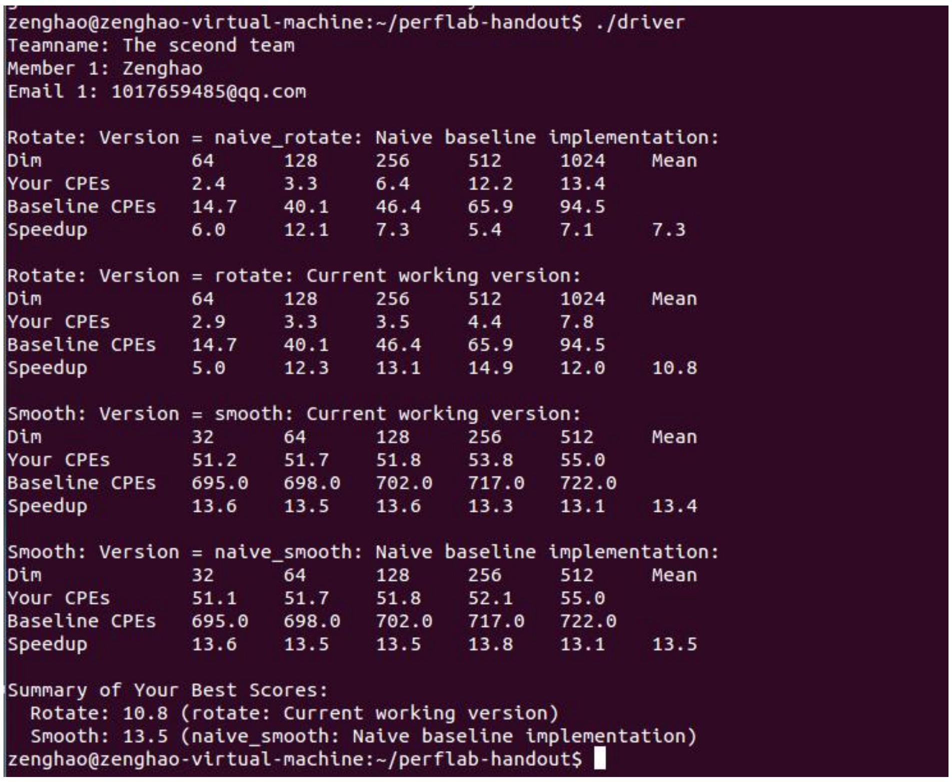
在循环分块时也要注意到分块的大小应能被测试 dim 整除。



```c
char rotate_descr[] = "rotate: Current working version";
void rotate(int dim, pixel *src, pixel *dst)
{
    int i,j,i1,j1;
    for (i1 = 0; i1< dim; i1+=4)
        for (j1 = 0; j1< dim; j1+=4)
                for(i=i1;i<i1+4;i++)
                        for(j=j1;j<j1+4;j++)
                                dst[RIDX(dim-1-j, i, dim)] = src[RIDX(i, j, dim)];
}
```

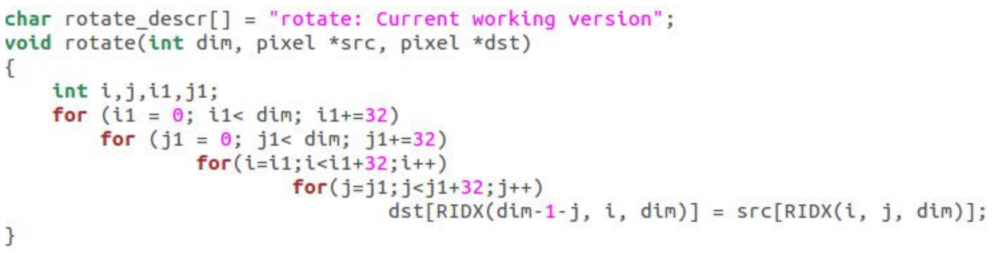
这个优化版本就把原循环分成了 4*4 小块，提高空间局部性。

测试结果：



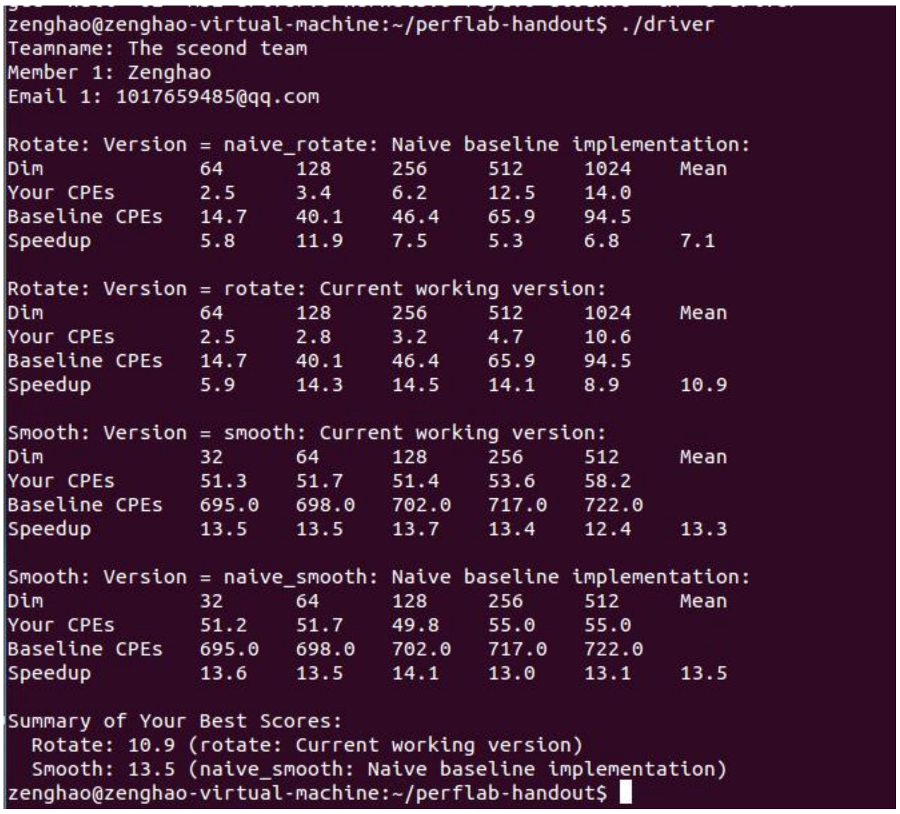
```
zenghao@zenghao-virtual-machine:~/perflab-handout$ ./driver
Teamname: The sceond team
Member 1: Zenghao
Email 1: 1017659485@qq.com

Rotate: Version = naive_rotate: Naive baseline implementation:
Dim             64      128     256     512     1024    Mean
Your CPEs       2.4     3.3     6.4     12.2    13.4
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         6.0     12.1    7.3     5.4     7.1     7.3

Rotate: Version = rotate: Current working version:
Dim             64      128     256     512     1024    Mean
Your CPEs       2.9     3.3     3.5     4.4     7.8
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         5.0     12.3    13.1    14.9    12.0    10.8

Smooth: Version = smooth: Current working version:
Dim             32      64      128     256     512     Mean
Your CPEs       51.2    51.7    51.8    53.8    55.0
Baseline CPEs   695.0   698.0   702.0   717.0   722.0
Speedup         13.6    13.5    13.6    13.3    13.1    13.4

Smooth: Version = naive_smooth: Naive baseline implementation:
Dim             32      64      128     256     512     Mean
Your CPEs       51.1    51.7    51.8    52.1    55.0
Baseline CPEs   695.0   698.0   702.0   717.0   722.0
Speedup         13.6    13.5    13.5    13.8    13.1    13.5

Summary of Your Best Scores:
  Rotate: 10.8 (rotate: Current working version)
  Smooth: 13.5 (naive_smooth: Naive baseline implementation)
zenghao@zenghao-virtual-machine:~/perflab-handout$
```

随后我尝试把循环分成 32*32 块：



```c
char rotate_descr[] = "rotate: Current working version";
void rotate(int dim, pixel *src, pixel *dst)
{
    int i,j,i1,j1;
    for (i1 = 0; i1< dim; i1+=32)
        for (j1 = 0; j1< dim; j1+=32)
                for(i=i1;i<i1+32;i++)
                        for(j=j1;j<j1+32;j++)
                                dst[RIDX(dim-1-j, i, dim)] = src[RIDX(i, j, dim)];
}
```

测试结果：



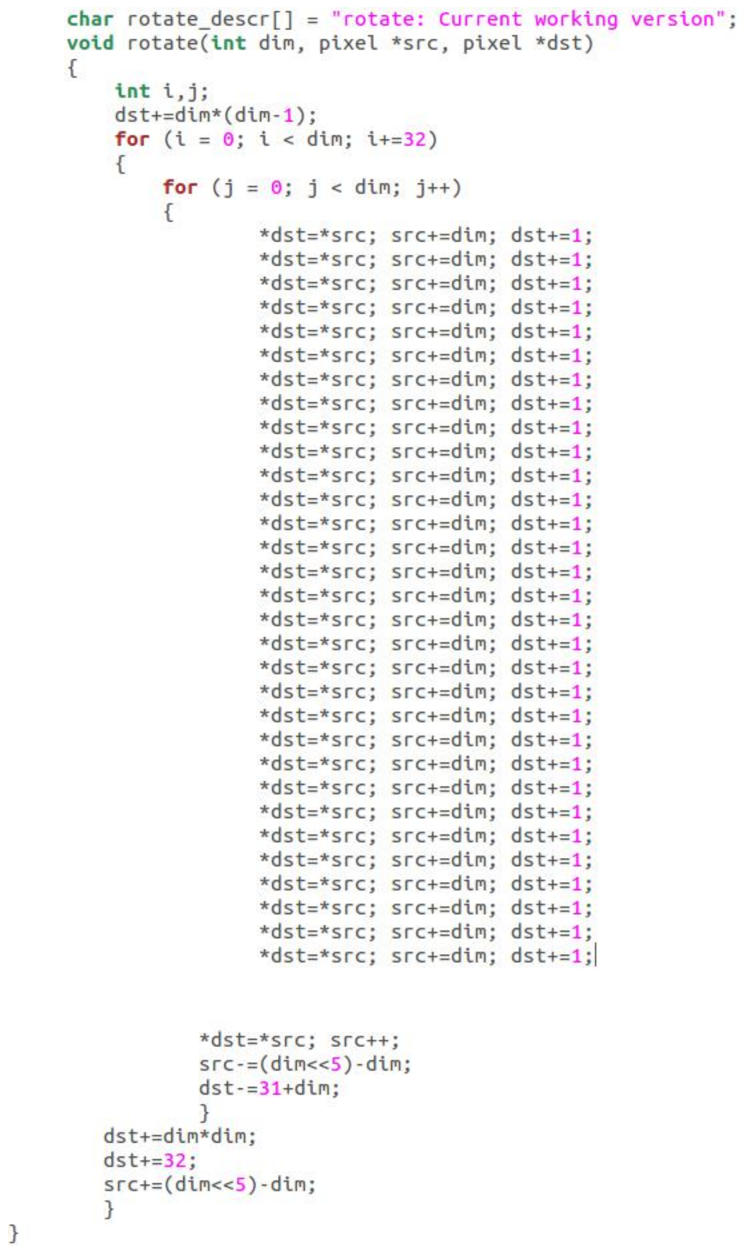
```
zenghao@zenghao-virtual-machine:~/perflab-handout$ ./driver
Teamname: The sceond team
Member 1: Zenghao
Email 1: 1017659485@qq.com

Rotate: Version = naive_rotate: Naive baseline implementation:
Dim             64      128     256     512     1024    Mean
Your CPEs       2.5     3.4     6.2     12.5    14.0
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         5.8     11.9    7.5     5.3     6.8     7.1

Rotate: Version = rotate: Current working version:
Dim             64      128     256     512     1024    Mean
Your CPEs       2.5     2.8     3.2     4.7     10.6
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         5.9     14.3    14.5    14.1    8.9     10.9

Smooth: Version = smooth: Current working version:
Dim             32      64      128     256     512     Mean
Your CPEs       51.3    51.7    51.4    53.6    58.2
Baseline CPEs   695.0   698.0   702.0   717.0   722.0
Speedup         13.5    13.5    13.7    13.4    12.4    13.3

Smooth: Version = naive_smooth: Naive baseline implementation:
Dim             32      64      128     256     512     Mean
Your CPEs       51.2    51.7    49.8    55.0    55.0
Baseline CPEs   695.0   698.0   702.0   717.0   722.0
Speedup         13.6    13.5    14.1    13.0    13.1    13.5

Summary of Your Best Scores:
  Rotate: 10.9 (rotate: Current working version)
  Smooth: 13.5 (naive_smooth: Naive baseline implementation)
zenghao@zenghao-virtual-machine:~/perflab-handout$
```

可以看到，与分成 4*4 小块后的结果相比速度比快了 0.1，因为 32*32 的分块充分利用了 cache 储存，则不能再继续分下去了，可能会超出 cache 储存。而故循环分块的优化效果只是一般，或许应该有更厉害的优化。

三、优化版本二

优化思路：循环展开

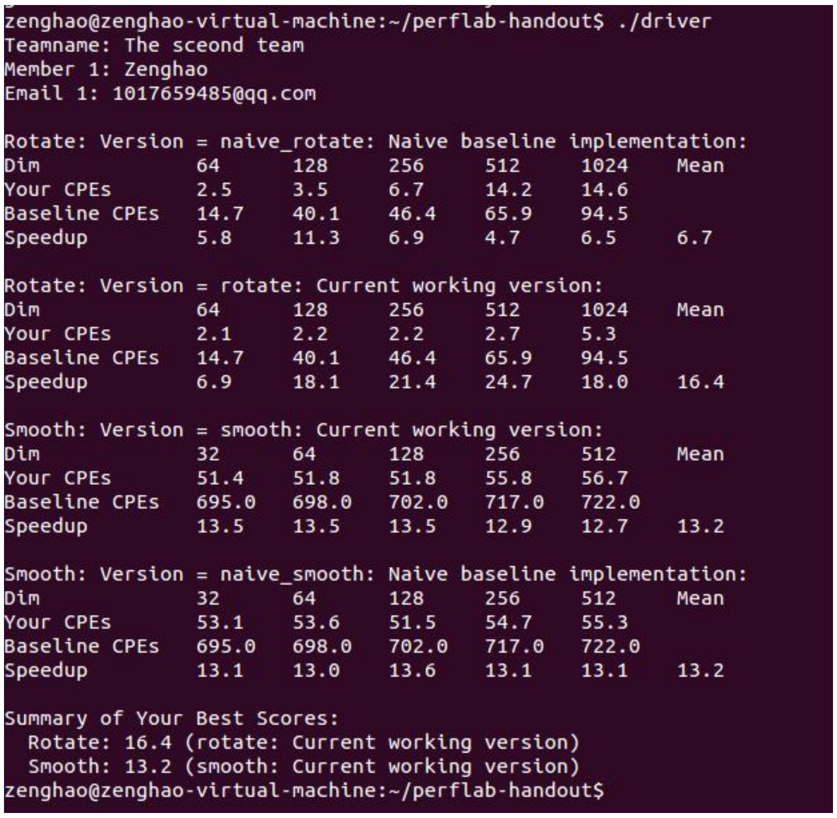
把循环展开，将循环次数减少了 32 倍，同时处理 32 个像素点来达到循环展开。32 个像素点并行处理，减少了关键路径的长度，有效提高了程序速度。



```
    char rotate_descr[] = "rotate: Current working version";
    void rotate(int dim, pixel *src, pixel *dst)
    {
        int i,j;
        dst+=dim*(dim-1);
        for (i = 0; i < dim; i+=32)
        {
            for (j = 0; j < dim; j++)
            {
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;
                    *dst=*src; src+=dim; dst+=1;


                *dst=*src; src++;
                src-=(dim<<5)-dim;
                dst-=31+dim;
                }
         dst+=dim*dim;
         dst+=32;
         src+=(dim<<5)-dim;
         }
}
```

优化代码即把矩阵第一列的 32 个元素转到最后一行的前 32 个元素的位置，再把第二行的 32 个元素转到倒数第二行的前 32 个元素的位置，依此进行下去，完成旋转操作。

测试结果：



```
zenghao@zenghao-virtual-machine:~/perflab-handout$ ./driver
Teamname: The sceond team
Member 1: Zenghao
Email 1: 1017659485@qq.com

Rotate: Version = naive_rotate: Naive baseline implementation:
Dim             64      128     256     512     1024    Mean
Your CPEs       2.5     3.5     6.7     14.2    14.6
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         5.8     11.3    6.9     4.7     6.5     6.7

Rotate: Version = rotate: Current working version:
Dim             64      128     256     512     1024    Mean
Your CPEs       2.1     2.2     2.2     2.7     5.3
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         6.9     18.1    21.4    24.7    18.0    16.4

Smooth: Version = smooth: Current working version:
Dim             32      64      128     256     512     Mean
Your CPEs       51.4    51.8    51.8    55.8    56.7
Baseline CPEs   695.0   698.0   702.0   717.0   722.0
Speedup         13.5    13.5    13.5    12.9    12.7    13.2

Smooth: Version = naive_smooth: Naive baseline implementation:
Dim             32      64      128     256     512     Mean
Your CPEs       53.1    53.6    51.5    54.7    55.3
Baseline CPEs   695.0   698.0   702.0   717.0   722.0
Speedup         13.1    13.0    13.6    13.1    13.1    13.2

Summary of Your Best Scores:
  Rotate: 16.4 (rotate: Current working version)
  Smooth: 13.2 (smooth: Current working version)
zenghao@zenghao-virtual-machine:~/perflab-handout$
```

将函数展开优化后的结果比原代码的结果速度比高了很多。