性能优化分析实验

常见代码优化方法

- 减少函数调用
- 提前计算
- 循环展开
- 并行运算
- 提高cache利用率

图像旋转

- 简单旋转

```
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)];
}
```

缺点：程序局部性不好，循环次数过多

第一次尝试：分块

- 尝试分成4*4的小块，提高空间局部性

```
void rotate(int dim, pixel *src, pixel *dst)
{
    int i,j,ii,jj;
    for(ii=0; ii < dim; ii+=4)
       for(jj=0; jj < dim; jj+=4)
           for(i=ii; i < ii+4; i++)
              for(j=jj; j < jj+4; j++)
                  dst[RIDX(dim-1-j,i,dim)] = src[RIDX(i,j,dim)];
}
```

测试CPE（cycles per element每元素周期数）改进为1.8

第二次尝试：循环展开

- 采用32*32分块，4*4路循环展开，注意循环内部语句执行顺序

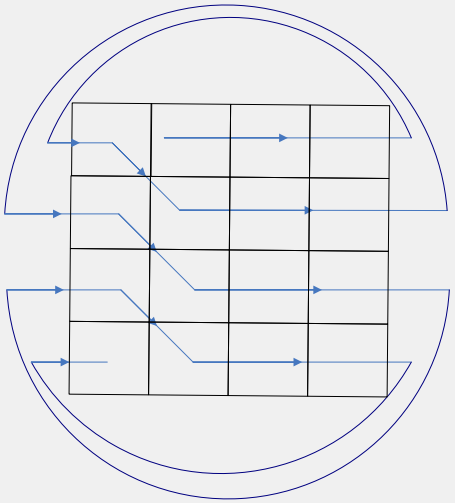
```
void rotate(int dim, pixel *src, pixel *dst)
{
    int i,j,ii,jj;
    for(ii=0; ii < dim; ii+=32)
       for(jj=0; jj < dim; jj+=32)
           for(i=ii; i < ii+32; i+=4)
              for(j=jj; j < jj+32; j+=4) {
                  dst[RIDX(dim-1-j,i,dim)] =     src[RIDX(i,j,dim)];
                  dst[RIDX(dim-1-j,i+1,dim)] =   src[RIDX(i+1,j,dim)];
                  dst[RIDX(dim-1-j,i+2,dim)] =   src[RIDX(i+2,j,dim)];
                  dst[RIDX(dim-1-j,i+3,dim)] =   src[RIDX(i+3,j,dim)];
                  dst[RIDX(dim-1-j-1,i,dim)] =   src[RIDX(i,j+1,dim)];
                  ...
}
```

测试CPE改进2.7

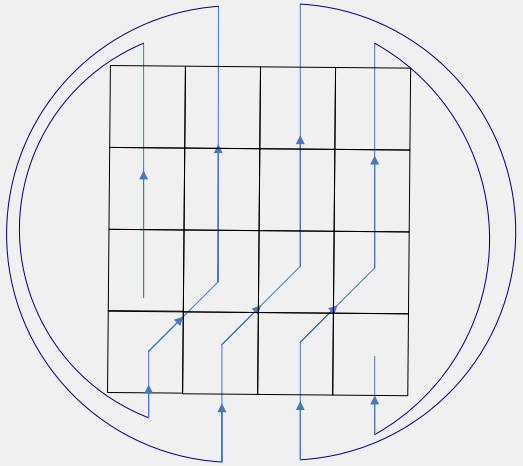
第三次尝试?

- 假设分块为4*4的矩阵，采用不同的巡回路线

src



dest



最后的尝试

- 考虑矩形分块32*1，32路循环展开，并使dest地址连续，以减少存储器写次数

```
#define COPY(d,s) *(d) = *(s)
void rotate(int dim, pixel *src, pixel *dst)
{
    int i, j;
    for (i = 0; i < dim; i+=32)
       for (j = dim-1; j >= 0; j-=1) {
            pixel *dptr = dst+RIDX(dim-1-j,i,dim);
            pixel *sptr = src+RIDX(i,j,dim);
            COPY(dptr, sptr); sptr += dim;
            COPY(dptr+1, sptr); sptr += dim;
            ...
            COPY(dptr+31, sptr);
           }
}
```

测得CPE改进为3.5

图像平滑

- naive_smooth

```
void naive_smooth(int dim, pixel *src, pixel *dst)
{   ...
    for (i = 0; i < dim; i++)
       for (j = 0; j < dim; j++)
           dst[RIDX(i, j, dim)] = avg(dim, i, j, src);
}

static pixel avg(int dim, int i, int j, pixel *src)
{
    ...
    initialize_pixel_sum(&sum);
    for(jj=max(j-1, 0); jj <= min(j+1, dim-1); jj++)
       for(ii=max(i-1, 0); ii <= min(i+1, dim-1); ii++)
           accumulate_sum(&sum, src[RIDX(ii,jj,dim)]);
    assign_sum_to_pixel(&current_pixel, sum);
    return current_pixel;
}
```

第一次尝试：减少函数调用

- 函数avg，accumulate_sum在smooth内实现
- 函数assign_sum_to_pixel，min，max用宏定义实现

```
#define fastmin(a,b) (a < b ? a : b)
#define fastmax(a,b) (a > b ? a : b)
```

得到的CPE 改进为1.6

第二次尝试：使用局部变量

- 让所有的函数调用均集成在smooth内，然后确定求平均值时涉及到的元素，它们的位置用4个局部变量记录下来，求和的像素个数用另外一个局部变量确定，最后做加法和除法

  得到的CPE 改进为5.0

第二次尝试：使用局部变量

```
void smooth_opt1(int dim, pixel *src, pixel *dst)
{   ...
    for(i = 0; i < dim; i++) {
          first = i*dim;
          test3 = (i-1 > 0 ? i-1 : 0);
          test4 = (i+1 < maxelement ? i+1 : maxelement);
          for(j = 0; j < dim; j++) {
              red = blue = green = 0;
              test1 = (j-1 > 0 ? j-1 : 0);
              test2 = (j+1 < maxelement ? j+1 : maxelement);
              num = (test2 - test1 + 1) * (test4 - test3 + 1);
              for(ii = test3; ii <= test4; ii++) {
                    second = ii*dim;
                    for(jj = test1; jj <= test2; jj++) {
                        temp2 = second + jj;
                        red += src[temp2].red;
               …} }
              temp1 = first + j;
              if(num == 9) {
                    dst[temp1].red = (unsigned short) (red/9);
                    ...
} }
```

第三次尝试：使用完全循环展开

- 分析不同的avg情况共有9种，4个角点位置，4个边带位置，1个中央块，因此只需对这9种情况分别讨论
- 比如左上角点

```
dst[0].red = (src[0].red + src[1].red + src[dim].red +
                src[dim + 1].red)/4;
dst[0].blue = (src[0].blue + src[1].blue +
                 src[dim].blue + src[dim + 1].blue)/4;
dst[0].green = (src[0].green + src[1].green +
          src[dim].green + src[dim + 1].green)/4;
```

第三次尝试：使用完全循环展开

- 中央位置

```
for(i = 1;i < dim-1; i++) {
    row = i*dim;
    for(j = 1;j < dim-1; j++) {
        dst[row + j].red = (src[row + j].red +
                            src[row + j - 1].red +
                            src[row + j + 1].red +
                            src[row + j - dim - 1].red +
                            src[row + j - dim].red +
                            src[row + j - dim + 1].red +
                            src[row + j + dim - 1].red +
                            src[row + j + dim].red +
                            src[row + j + dim + 1].red)/9;
    ...
}
```

最终测得CPE改进为8.6

最后尝试?

- 减少冗余计算

  预先确定操作数的地址，使用指针变量

  s0=src+row

  s1=src+row-dim

  s2=src+row+dim

- 变常数除法为乘法

  /4等效成>>2

  /6等效*0x2AAB>>16

  /9等效成*0x1C72>>16

推荐优化代码相关书籍

- Write Great Code Volume 2: Thinking Low-level，Writing High-level --- Randall Hyde

- Software Optimization for High-Performance Computing: Creating Faster Applications
  ---K. P.Wadleigh and I. L. Crawford.