perflab1 实验日志

一、优化方法

不依赖于目标机器的特性的优化方法：

5.4　消除循环的低效率

优化主要体现在消除循环内被多次重复计算的变量、测试条件（这些值在循环中不会改变）从而减少 CPU 计算次数。书上介绍的优化（也是一种常用的优化）——代码移动，这类优化包括识别要执行多次（例如在循环里）但是计算结果不会改变的计算，随后将计算移动到代码前面不会被多次求值的部分，通常用于计算较多且重复性较大的代码优化。

5.5　减少过程调用

过程的调用涉及多级结构，注定会为程序带来相当大的开销，并且妨碍大多数形式的优化。因此，我们可以最大程度上的减少过程调用，采取更为直观、直接的方法来取代过程调用（通常体现为直接实现过程的功能）的功能从而减少过程调用、减少开销。但这样注定会损害程序的模块性与抽象性——减少过程调用我们就必须知道该过程的具体实现并取而代之，从而损害模块性与抽象性。因此，该优化需要程序员在性能与模块性、抽象性作出权衡且该部分带来的优化效果通常较低。

5.6　消除不必要的寄存器引用

优化主要体现在减少不必要的寄存器的读写以提高程序性能(往往能显著提高性能)。不过必须保证的是保持程序的正确性前提下可变换。但是个人认为该优化更贴近计算机的底层行为，面向的更多是汇编语言层面，如果在高级语言上进行操作，可行性较低。所以，该优化更贴近底层，需要程序员具备一定的汇编知识以及对计算机顶层行为有一定了解。

二、说明

Part A——rotate 函数的作用：用于将图像逆时针选择 90°　实现过程图示

将图像看成由像素组成的二维数组矩阵，实现过程主要为两部分——转置后行交换。

kernels.c 代码中函数、数据结构的定义在 defs.h 中体现

三、优化实现

原始版本代码

```
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)];
}
```

优化一：5.4 消除循环的低效率

```
void rotate1(int dim, pixel *src, pixel *dst)
{
    int i, j, temp; //设置一个中间变量tmp，用来存储中间值
    for (j = 0; j < dim; j++)
    {
        temp = dim - 1 - j; //由于dim-1-j会经常使用，所以这里进行提前计算，省去了每次计算的时间
        for (i = 0; i < dim; i++)
            dst[RIDX(temp, i, dim)] = src[RIDX(i, j, dim)]; //这里不再对dim-1-j再进行运算
    }
}
```

优化一文字描述：

这里是书上的 5.4 消除循环的低效率。通过观察源代码可以发现在循环中对 dim-1-j 这个数据进行了重复的计算，所以这里可以对循环进行修改，将计算时的行列进行调换，可以提前计算 dim-1-j，这样就可以省去每一次循环中重复计算的时间，可以很好的提高效率。

性能测试：

```
Rotate: Version = naive_rotate: Naive baseline implementation:
Dim             64      128     256     512     1024    Mean
Your CPEs       1.4     2.0     3.4     6.9     6.8
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         10.3    20.2    13.6    9.6     14.0    13.1

Rotate: Version = rotate1: Current working version:
Dim             64      128     256     512     1024    Mean
Your CPEs       1.3     1.4     1.8     2.7     3.9
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         11.5    29.2    25.7    24.8    24.4    22.0
```

优化二：5.5 减少过程调用

```
void rotate2(int dim, pixel *src, pixel *dst)
{
    int i, j;
    for (i = 0; i < dim; i++)
        for (j = 0; j < dim; j++)
            dst[(dim - 1 - j) * dim + i] = src[i * dim + j]; //省去调用函数的时间
}
```

优化二文字描述：

这是书上的 5.5 减少过程调用，书上说可以通过消除函数调用来提高效率，我看了一下，在循环过程中重复调用了一个 RIDX 函数，这个函数在 defs.h 中被宏定义，它的功能就是计算三个参数 i,j,n 的参数式 i*n+j 的数值，

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

所以这里我可以直接将其替换为 i*n+j,而省去了调用函数的时间，但总体上优化不明显。

性能测试：

```
Rotate: Version = naive_rotate: Naive baseline implementation:
Dim             64      128     256     512     1024    Mean
Your CPEs       1.4     1.9     3.5     7.2     6.8
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         10.6    20.8    13.3    9.2     13.8    13.0

Rotate: Version = rotate2: Current working version:
Dim             64      128     256     512     1024    Mean
Your CPEs       1.4     1.9     3.4     7.0     6.8
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         10.6    20.9    13.6    9.5     13.9    13.2
```

优化三：增加循环内每步操作（同时结合 5.4、5.5）

```
void rotate3(int dim, pixel *src, pixel *dst)
{
    int i, j;
    for (i = 0; i < dim; i=i+2) //步长调整为2
        for (j = 0; j < dim; j=j+2) //步长调整为2
        {
            dst[RIDX(dim - 1 - j, i, dim)] = src[RIDX(i, j, dim)];
            dst[RIDX(dim - 2 - j, i, dim)] = src[RIDX(i, j+1, dim)];
            dst[RIDX(dim - 1 - j, i+1, dim)] = src[RIDX(i+1, j, dim)];
            dst[RIDX(dim - 2 - j, i+1, dim)] = src[RIDX(i+1, j+1, dim)];
        }
}
```

优化三文字描述：

这个方法是通过在 for 循环中尽可能多的进行操作，以此来达到提高效率的目的，这里我只是将步长提高到了 2，可以将步长提高到 2 的倍数，直到 32。

性能测试：

```
Rotate: Version = naive_rotate: Naive baseline implementation:
Dim             64      128     256     512     1024    Mean
Your CPEs       1.4     2.0     3.5     7.0     6.9
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         10.6    20.5    13.2    9.4     13.7    13.0

Rotate: Version = rotate3: Current working version:
Dim             64      128     256     512     1024    Mean
Your CPEs       1.3     1.8     2.5     3.9     6.9
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         10.9    21.8    18.6    16.9    13.6    15.9
```

优化四：将循环展开（增加循环步长的更高版本，以 32 为一步长，同时结合 5.4、5.5、5.6）

```c
void rotate4(int dim, pixel *src, pixel *dst)
{
    int i, j, tmp1 = dim * dim, tmp2 = dim * 31, tmp3 = tmp1 - dim, tmp4 = tmp1 + 32, tmp5 = dim + 31; //定义中间变量
    dst += tmp3;
    for (i = 0; i < dim; i += 32)
    {
        for (j = 0; j < dim; j++) //将for循环展开，并通过指针的形式实现对图形的翻转，以32为一个步长
        {
            *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;
```

```c
            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;
```

```c
            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 += d pixel *src
            *dst = *src;
            dst++;
            src += dim;
            *dst = *src;
            dst++;
            src += dim;
            *dst = *src;
            dst++;
            src += dim;
            *dst = *src;
            src++;
            src -= tmp2;
            dst -= tmp5;
        }
        src += tmp2;
        dst += tmp4;
    }
}
```

优化四文字描述：

这里运用了增加循环内操作步数的极限思想，将循环内每次循环操作步数扩充到了 32 步，同时结合优化 1 定义中间变量减少重复计算消除低效率、结合优化 2 减少过程调用，结合 5.6 优化思路，使用指针进行赋值操作从而不需要额外的用寄存器来保存像素的位置，不需要再为这部分寄存器进行读写，从而减少了对不必要寄存器的读写操作。此优化为优化的最终版本，结合了 5.4、5.5、5.6 所有的优化思想（也可以更加继续 1，将步长继续增加）。

性能测试：

```
Rotate: Version = naive_rotate: Naive baseline implementation:
Dim             64      128     256     512     1024    Mean
Your CPEs       1.4     2.0     3.7     7.1     6.8
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         10.2    20.2    12.4    9.2     13.8    12.7

Rotate: Version = new_rotate3: Current working version:
Dim             64      128     256     512     1024    Mean
Your CPEs       1.4     1.6     1.9     2.4     3.5
Baseline CPEs   14.7    40.1    46.4    65.9    94.5
Speedup         10.6    25.1    24.5    27.6    27.2    21.8
```