第六章

6.42

因为高速缓存每行有4个字节，而`pixel`中有4个`char`类型数据，刚好为4字节，所以每次访问`buffer[i][j].r`都会出现不命中，从而将`buffer[i][j].r` `buffer[i][j].g` `buffer[i][j].b` `buffer[i][j].a`加载到缓存中，访问`buffer[i][j].g` `buffer[i][j].b` `buffer[i][j].a`时会直接命中，因此不命中率为25%

6.43

`cptr`指针被初始化为`buffer`数组的首地址，在不超过数组最后一个地址的情况下，每次把地址+1（即访问下一个元素），并把其中的数据设为0，和上题一样，因为`pixel`中有4个`char`类型数据，刚好为4字节，所以每次访问`buffer[i][j].r`都会出现不命中，从而将`buffer[i][j].r` `buffer[i][j].g` `buffer[i][j].b` `buffer[i][j].a`加载到缓存中，访问`buffer[i][j].g` `buffer[i][j].b` `buffer[i][j].a`时会直接命中，因此不命中率为25%

6.44

`cptr`指针被初始化为`buffer[0][0]`的首地址，因为`int`类型的指针会指向4个字节的首地址，故而，在不超过数组最后一个地址的情况下，每次把地址+4（即访问下一个元素），再会把`cptr`指针指向的4个字节初始化为`int`类型的0，而高速缓存每行有4个字节，因此每次都会把刚刚访问的4个字节加载到高速缓存中，从而导致每次访问都不命中，因此不命中率为100%

6.46

```
void transpose(int *dst, int *src, int dim, int blk) //dim为矩阵的维度  blk为转置操作的块大小
{
  int bc, br, bcn, brn, i, j;
  /*
    br: 当前处理的行块的起始行
    bc: 当前处理的列块的起始列
    bcn: 当前处理的列块的终止列
    brn: 当前处理的行块的终止行
  */
  br = 0;
  do {
    brn = min(blk + br, dim); //确定当前行块的终止行
    // printf("br %d, brn %d\n", br, brn);
    bc = 0;
    do {
      bcn = min(blk + bc, dim); // 确定当前列块的终止列
      // printf("bc %d, bcn %d\n", bc, bcn);

      // 对当前块中的元素进行转置
      for (i = br; i < brn; ++i) {
        for (j = bc; j < bcn - 1; j+=2) {
        // it turns out loop unrolling does not help
```

```
          dst[j*dim + i] = src[i*dim + j]; //将输入矩阵中第 i 行第 j 列的元素复制到输出矩阵中第 j 行第 i 列
          dst[(j+1)*dim + i] = src[i*dim + j + 1]; //将输入矩阵中第 i 行第 j+1 列的元素复制到输出矩阵中第 j+1 行第 i 列
        }
         // 处理当前块中剩余的元素
        for (; j < bcn; ++j)
          dst[j*dim + i] = src[i*dim + j];
      }

      bc = bcn; // 更新列块的起始列
    } while (bc < dim);

    br = brn; // 更新行块的起始行
  } while (br < dim);
}
```

该函数实现了矩阵转置操作，将一个大小为 `dim` × `dim` 的二维数组 `src` 转置后存储到另一个同样大小的二维数组 `dst` 中。转置操作涉及到交换矩阵的行和列，使得 `src` 中位置为 `(i,j)` 的元素在 `dst` 中变为位置为 `(j,i)` 的元素。

该函数还有两个额外的参数 `blk` 和 `dim`，用于确定转置操作的块大小。函数使用嵌套循环以块大小 `blk` 遍历矩阵元素，以减少缓存未命中并提高性能。

外层循环以块大小 `blk` 按行遍历矩阵，而内层循环以块大小 `blk` 按列遍历矩阵。