SIMD

SIMD是（Single Structure Multi Data）的简称，一般是在硬件上实现。简单来说，就是将多个数据当作一个特定的结构去处理，这样可以极大地提升大规模数据的处理速度。

下面以一个简单的图像处理为例子：

对于BMP格式的图像，在二进制文件中RGB颜色以BGR的顺序存储，一般每个分量的颜色大小为8 bits，即可以储存一个从0到255的无符号整数，而一个像素包含三个分量的颜色，则一个像素的大小则为24 bits。下面，我们假设一张 100 * 100 的RGB图像以BMP格式存储，并且其图像数据被写入一个大小为 100*100*3 byte 的缓存，下面以c++代码形式展示：

```
unsigned char * buffer = new unsigned char[30000];
readBMPImage(buffer, "source.bmp");
// buffer = { B_0_0, G_0_0, R_0_0, B_0_1, G_0_1, R_0_1, ... , B_99_99, G_99_99, R_99_99 }
```

但是，对于大部分习惯使用RGB顺序的程序员以及他们写出的图形接口而言，BGR顺序并不受欢迎。那么怎么办呢？自然是将BGR颜色转换为RGB颜色，下面给出一个单线程下的一般做法：

```
for (int i = 0; i < 30000; i += 3)
{
    std::swap(buffer[i], buffer[i+2]);
}
```

下面，我们用SIMD方法来转换颜色空间：

```
union {
    unsigned char c[16];
    __uint128_t   i;
} mask = {
    .c = {0xFF, 0x00, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00} };
// the step above generated a mask that looks like this in binary:
// 11111111 11111111 00000000 00000000 00000000 00000000 11111111 11111111 ...
__uint128_t s0, sr, sg, sb;

for (int i = 0; i < 30000; i += 15)
{
    memcpy(&s0, buffer + i, 15);
    sr = mask.i & s0;         // extract R channels by using bit operator
                              //
    sg = mask.i & (s0 >> 8);  // suppose digits are stored as 'big endian' order
                              // that a 2byte integer 258 = 256 + 2 = 0x0102
                              // is stored in memory as 00000010 00000001 or 0x02 0x01
```

```
                                //
                                // therefore, a right shift operator is actually performed
                                // by left shift in memory, and by left shift s0, we align
                                // G channels to the mask
                                //
    sb = mask.i & (s0 >> 16);
    s0 = std::move(sb);         // faster than memcpy
    s0 |= (sr << 16);
    s0 |= (sg << 8);
    memcpy(buffer + i, &s0, 15);
    buffer_ptr += 15;
}
```

可以看到，使用一个 128 bits 的大整数结构可以一次性处理5个像素的转换，而且由于在硬件层面，这个128 bits的处理和以一个32 bits的整数的处理速度是一样的，因此，在同样是单线程的颜色转换任务上，SIMD可以将运行时间缩减到 1/3 到 1/5 左右，极大提升了运算性能。