浙江大学实验报告

课程名称：汇编与接口

实验项目名称：**对向量化指令的探索与实验分析**

实验日期：2021.12.30

背景说明

SIMD全称Single Instruction Multiple Data，单指令多数据流，能够复制多个操作数，并它们打包在大型寄存器的一组指令集。使用SIMD指令，可以同时在多个操作数上执行同一指令，进而加快运行速度。举个例子，一般SISD（Single Instruction Single Data）指令，如多个加法指令，会依次访问内存、取得操作数并相加；而如果使用SIMD指令，程序则会为这几条加法指令同时访问内存取得操作数，再同时运行加法运算，加快了运行速度。

SIMD的首次使用出现在1966年。SIMD是1980年代早期矢量超级计算机的基础。事实上，从SIMD的原理不难看出矢量化的影子。而现代SIMD计算机的第一个时代以大规模并行处理风格的超级计算机为特征。

到了当下，当前的SIMD指令被广泛用于计算机市场，而不限于超级计算机市场。SIMD的一大应用是游戏、音频、视频的处理，这种多媒体的处理尤其适合并行计算。1996年，英特尔在x86架构内部署MMX扩展，是为第一个广泛部署的台式机SIMD。此后，英特尔又逐步开发出了SSE、AVX等SIMD指令集。这三种指令集的简介如下：

- MMX是最早的SIMD指令集，其使用过程占用八个浮点数寄存器；数据宽度需要对齐
- SSE在MMX的基础上发展而来，英特尔为其特地设置了单独的128位XMM寄存器，使得其执行不再与浮点运算相矛盾；引入了SIMD浮点数据类型
- AVX在SSE的基础上发展而来，将128位的XMM寄存器扩展为了256位的YMM寄存器

其发展历史如下：

IS	Number	Date	ICPU	IDate	ACPU	ADate
MMX	57	1996-10-12	Pentium MMX(P55C)	1996-10-12	K6	1997-4-1
SSE	70	1999-5-1	Pentium III(Katmai)	1999-5-1	Athlon XP	2001-10-9
SSE2	144	2000-11-1	Pentium 4(Willamette)	2000-11-1	Opteron	2003-4-22
SSE3	13	2004-2-1	Pentium 4(Prescott)	2004-2-1	Athlon 64	2005-4-1
SSSE3	16	2006-1-1	Core	2006-1-1	Fusion(Bobcat)	2011-1-5
SSE4.1	47	2006-9-27	Penryn	2007-11-1	Bulldozer	2011-9-7
SSE4.2	7	2008-11-17	Nehalem	2008-11-17	Bulldozer	2011-9-7
SSE4a	4	2007-11-11			K10	2007-11-11
SSE5		2007-8-30				
AVX		2008-3-1	Sandy Bridge	2011-1-9	Bulldozer	2011-9-7
AVX2		2011-6-13	Haswell	2013-4-1		
AES	7	2008-3-1	Westmere	2010-1-7	Bulldozer	2011-9-7
3DNowPrefetch	2	2010-8-1			K6-2	1998-5-28
3DNow!	21	1998-1-1			K6-2	1998-5-28
3DNow!+		1999-6-23			Athlon	1999-6-23
MmxExt					Athlon	1999-6-23
3DNow! Pro					Athlon XP	2001-10-9
POPCNT	1	2007-11-11			K10	2007-11-11
ABM	1	2007-11-11			K10	2007-11-11
CLMUL	5	2008-5-1	Westmere	2010-1-7	Bulldozer	2011-9-7

IS	Number	Date	ICPU	IDate	ACPU	ADate
F16C		2009-5-1	Ivy Bridge	2012-4-1	Bulldozer	2011-9-7
FAM4		2009-5-1			Bulldozer	2011-9-7
XOP		2009-5-1			Bulldozer	2011-9-7

SIMD适合用于多媒体处理，由此我想到了利用它来进行BMP图片处理。位图图像（bitmap），亦称为点阵图像或栅格图像，是由称作像素（图片元素）的单个点组成的。这些点可以进行不同的排列和染色以构成图样。位图的特点是可以表现色彩的变化和颜色的细微过渡，产生逼真的效果，缺点是在保存时需要记录每一个像素的位置和颜色值，占用较大的存储空间。实验中用到的`bmp.h`和`bmp.cpp`均是自主编写。

探索过程

我决定通过对比来体现SIMD的优势，探索的方法如下：

1. 计时比较SIMD与SISD的速度差别、编译器优化和SIMD intrinsic指令的速度差别，在Linux系统下进行探索可以得到更准确的计时结果
2. 选用的SIMD指令集为SSE
3. 编写C++程序进行BMP格式图片的灰度化，编写时准备一个普通的灰度化函数以及一个含有SIMD intrinsic指令的灰度化函数
4. 使用普通函数时，对编译器指定不同的优化等级，对比开启SIMD和不开启SIMD的速度差别
5. 对编译器指定开启向量化，使用普通函数和intrinsic函数分别运行，对比编译器优化和使用intrinsic指令的速度差别

首先我查看我的设备是否支持SIMD，在控制台输入`cat /proc/cpuinfo`命令，可以看到设备支持mmx、sse、sse2等SIMD指令集。

然后，查看gcc是否支持SIMD，在控制台输入`gcc -march=native -c -Q --help=target`命令，可以看到gcc同样支持指定mmmx、mavx、msse等参数。

确认以上两步后，开始编写代码。考虑到SIMD指令适合用于多媒体处理的场景，这一次我决定用图片处理来进行探索实验。原理非常简单，读入BMP图片后，对每个像素的RGB做一定的计算再输出，将整个图片变灰。其中最关键的一步计算是利用公式$grey = (R * 77 + G * 151 + B * 28)/256$来得到像素的灰度。为了方便SIMD计算，在读入像素的RGB时我将像素的RGB分别存储到三个数组当中，并且将8位的`unsigned char`数据类型转化为`int`，在保存到新的BMP文件之前再把它转化为`unsigned char`。

```
int main(){
    // definition
    FILE *source, *dest;
    Palette *pa = new Palette[256];
    clock_t start, end;
    ...
    // open file
    if((source = fopen("./soulmate.bmp", "rb")) == NULL){
        cout << "Opening fails!" << endl;
        return -1;
    }
    if((dest = fopen("./soulmate_grey.bmp", "wb")) == NULL){
```

```
        cout << "Opening fails!" << endl;
        fclose(source);
        return -1;
    }

    // read in source file header
    loadBMPheader(source, header);
    length = header.width * header.height;
    ...
    // read in source file data
    loadBMPdata(source, length, R, G, B);
    fclose(source);

    // redefine header
    // redefine information
    header.colorBits = 8;
    header.dataOffset = sizeof(Header) + 256*sizeof(RGBQ);
    header.dataSize = length;
    header.fileSize = header.dataSize + header.dataOffset;
    header.resolution[0] = header.resolution[1] = 0;

    start = clock();
    // calculate grey degree
    // calGrey_sse();
    calGrey();
    end = clock();

    // save BMP
    for(int i=0; i<length; i++){
        destdata[i] = (unsigned char)dstdata[i];
    }
    saveBMP(dest, header, pa, destdata, length);
    ...
    fclose(dest);
    cout << "Time for calculation: " << (double)(end - start)/CLOCKS_PER_SEC <<
endl;
    return 0;
}
```

一个仅仅通过循环实现的灰度计算函数如下，简单地通过循环计算公式即可：

```
// grey = (R * 77 + G * 151 + B * 28 ) / 256
void calGrey(){
    for(int i = 0; i<length; i++){
        dstdata[i] = ((R[i] * 77 + G[i] * 151 + B[i] * 28 ) / 256);
    }
    return ;
}
```

通过调用intrinsic指令实现的灰度计算函数如下。首先计算一个128位的块能够包含4个`int`型的元素。经过探索，似乎SSE并没有元素长度为一个字节的并行计算函数，这也是为什么我在读入BMP时要把RGB的数据类型保存为`int`型。然后计算需要有多少个块需要循环处理，以及剩余的元素有多少个。对可以循环计算的部分，循环调用intrinsic函数进行计算，并及时保存结果。对于剩余的部分，在循环结束后单独处理。经过探索，SSE似乎也并没有整数除法的并行函数，因此在每个循环的最后我只能串行实现公式的除法部分。（这里不小心把151打成了152，但是不影响结果。）

```
// grey = (R * 77 + G * 151 + B * 28 ) / 256
void calGrey_sse()
{
    __attribute__((aligned(32))) int arr77[4] = {77, 77, 77, 77};
    __attribute__((aligned(32))) int arr152[4] = {152, 152, 152, 152};
    __attribute__((aligned(32))) int arr28[4] = {28, 28, 28, 28};
    size_t i, block = 4;    // 128/32 = 4
    size_t numBlock = length / block;   // number blocks
    size_t numRem = length % numBlock;  // remainder
    __m128i rR, rG, rB, rRes, rmid;
    // pointers
    const __m128i* rptr = (const __m128i*)R;
    const __m128i* gptr = (const __m128i*)G;
    const __m128i* bptr = (const __m128i*)B;
    __m128i* bufptr = (__m128i*)dstdata;
    __m128i r77 = _mm_load_si128((const __m128i*)arr77);
    __m128i r152 = _mm_load_si128((const __m128i*)arr152);
    __m128i r28 = _mm_load_si128((const __m128i*)arr28);
    int32_t *bufp = (int32_t *)bufptr;

    // calculation
    for(i=0; i<numBlock; i++)
    {
        rRes = _mm_setzero_si128();
        rR = _mm_load_si128(rptr);
        rG = _mm_load_si128(gptr);
        rB = _mm_load_si128(bptr);
        rmid = _mm_mullo_epi32(rR, r77);
        rRes = _mm_add_epi32(rRes, rmid);
        rmid = _mm_mullo_epi32(rG, r152);
        rRes = _mm_add_epi32(rRes, rmid);
        rmid = _mm_mullo_epi32(rB, r28);
        rRes = _mm_add_epi32(rRes, rmid);
        _mm_store_si128(bufptr, rRes);
        bufp = (int32_t *)bufptr;
        bufp[0] /= 256;
        bufp[1] /= 256;
        bufp[2] /= 256;
        bufp[3] /= 256;
        bufptr++;
        rptr++;
        gptr++;
        bptr++;
    }

    // deal with remainder
    int32_t *rp = (int32_t *)rptr;
    int32_t *gp = (int32_t *)gptr;
    int32_t *bp = (int32_t *)bptr;
    bufp = (int32_t *)bufptr;
    for(i=0; i<numRem; i++)
    {
        bufp[i] = (rp[i]*77 + gp[i]*152 + bp[i]*28)/256;
    }

}
```

经过验证，对于一个800*800的图片输入：

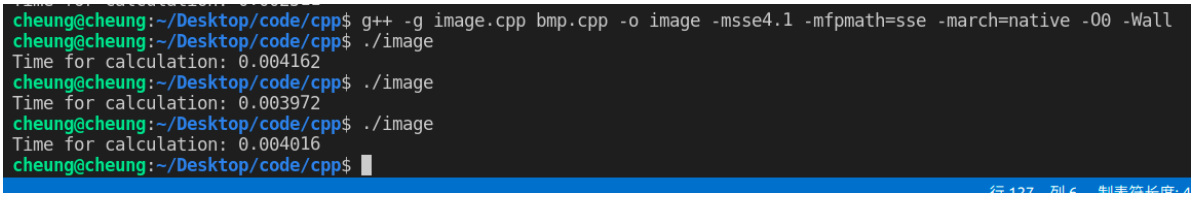
两个函数均可以正确输出灰度化后的图片：

结果分析

在实验前，我本来还是希望在一个不开启向量化的优化等级（O0到O2）下，开启向量化进行对比。但是我尝试了若干种gcc的参数设置都无法实现这个效果。然后我搜索了一些资料，发现我看过的所有资料都是同时指定`-O3`参数和SIMD指令集参数。于是我也采用了类似的参数设置。

普通循环无向量化

指定参数为 `g++ -g image.cpp bmp.cpp -o image -msse4.1 -mfpmath=sse -march=native -O0 -Wall`，可以看到灰度化的计算时间约为0.004秒。



查看函数`calGrey()`的汇编代码如下：

```
   0x00000000000012e9 <+0>:     endbr64
   0x00000000000012ed <+4>:     push   %rbp
   0x00000000000012ee <+5>:     mov    %rsp,%rbp
   0x00000000000012f1 <+8>:     movl   $0x0,-0x4(%rbp)
   0x00000000000012f8 <+15>:    mov    -0x4(%rbp),%eax
   0x00000000000012fb <+18>:    movslq %eax,%rdx
   0x00000000000012fe <+21>:    mov    0x3f0b(%rip),%rax        # 0x5210
<length>
   0x0000000000001305 <+28>:    cmp    %rax,%rdx
   0x0000000000001308 <+31>:    jge    0x138f <calGrey()+166>
   0x000000000000130e <+37>:    mov    0x3e8b(%rip),%rax        # 0x51a0 <R>
   0x0000000000001315 <+44>:    mov    -0x4(%rbp),%edx
   0x0000000000001318 <+47>:    movslq %edx,%rdx
   0x000000000000131b <+50>:    shl    $0x2,%rdx
   0x000000000000131f <+54>:    add    %rdx,%rax
   0x0000000000001322 <+57>:    mov    (%rax),%eax
   0x0000000000001324 <+59>:    imul   $0x4d,%eax,%edx
   0x0000000000001327 <+62>:    mov    0x3e92(%rip),%rax        # 0x51c0 <G>
   0x000000000000132e <+69>:    mov    -0x4(%rbp),%ecx
   0x0000000000001331 <+72>:    movslq %ecx,%rcx
   0x0000000000001334 <+75>:    shl    $0x2,%rcx
   0x0000000000001338 <+79>:    add    %rcx,%rax
   0x000000000000133b <+82>:    mov    (%rax),%eax
   0x000000000000133d <+84>:    imul   $0x97,%eax,%eax
   0x0000000000001343 <+90>:    lea    (%rdx,%rax,1),%ecx
   0x0000000000001346 <+93>:    mov    0x3e93(%rip),%rax        # 0x51e0 <B>
   0x000000000000134d <+100>:   mov    -0x4(%rbp),%edx
   0x0000000000001350 <+103>:   movslq %edx,%rdx
   0x0000000000001353 <+106>:   shl    $0x2,%rdx
   0x0000000000001357 <+110>:   add    %rdx,%rax
   0x000000000000135a <+113>:   mov    (%rax),%eax
   0x000000000000135c <+115>:   imul   $0x1c,%eax,%eax
   0x000000000000135f <+118>:   add    %eax,%ecx
   0x0000000000001361 <+120>:   mov    0x3e98(%rip),%rax        # 0x5200
<dstdata>
   0x0000000000001368 <+127>:   mov    -0x4(%rbp),%edx
   0x000000000000136b <+130>:   movslq %edx,%rdx
   0x000000000000136e <+133>:   shl    $0x2,%rdx
   0x0000000000001372 <+137>:   add    %rax,%rdx
   0x0000000000001375 <+140>:   mov    %ecx,%eax
   0x0000000000001377 <+142>:   lea    0xff(%rax),%ecx
   0x000000000000137d <+148>:   test   %eax,%eax
   0x000000000000137f <+150>:   cmovs  %ecx,%eax
   0x0000000000001382 <+153>:   sar    $0x8,%eax
```

```
   0x0000000000001385 <+156>:   mov    %eax,(%rdx)
   0x0000000000001387 <+158>:   incl   -0x4(%rbp)
   0x000000000000138a <+161>:   jmpq   0x12f8 <calGrey()+15>
   0x000000000000138f <+166>:   nop
   0x0000000000001390 <+167>:   pop    %rbp
   0x0000000000001391 <+168>:   retq
```

这一串汇编代码非常清晰，一个SIMD指令都没有。可以看到`-0x4(%rbp)`存放着循环用的变量`i`，也可以从汇编代码清晰地看出汇编代码在按部就班地执行：先获取`R[i]`并乘上79，加上`G[i]`乘151和`B[i]`乘28。获得`dstdata[i]`的地址后，通过右移8位实现除以256的效果，把计算结果存入`datdata[i]`。`jmpq`控制的循环结构也非常明显。

普通循环向量化

指定优化等级为三级`g++ -g image.cpp bmp.cpp -o image -msse4.1 -mfpmath=sse -march=native -O3 -Wall`，可以看到执行时间约为0.0015秒。

```
cheung@cheung:~/Desktop/code/cpp$ g++ -g image.cpp bmp.cpp -o image -msse4.1 -mfpmath=sse -march=native -O3 -Wall
bmp.cpp: In function 'void loadBMPheader(FILE*, Header&)':
bmp.cpp:6:10: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unu
sed_result [-Wunused-result]
    6 |     fread(&header, 1, 54, file);
      |     ~~~~~^~~~~~~~~~~~~~~~~~~~~~
bmp.cpp: In function 'void loadBMPdata(FILE*, long int, int*, int*, int*)':
bmp.cpp:32:14: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_un
used_result [-Wunused-result]
   32 |         fread(rgb, 1, 3, file);
      |         ~~~~~^~~~~~~~~~~~~~~~~
cheung@cheung:~/Desktop/code/cpp$ ./image
Time for calculation: 0.001545
cheung@cheung:~/Desktop/code/cpp$ ./image
Time for calculation: 0.001564
cheung@cheung:~/Desktop/code/cpp$ ./image
Time for calculation: 0.001498
cheung@cheung:~/Desktop/code/cpp$
```

查看函数`calGrey()`的汇编代码如下：

```
   0x0000000000001720 <+0>:     endbr64
   0x0000000000001724 <+4>:     mov    0x3a35(%rip),%rcx        # 0x5160
<length>
   0x000000000000172b <+11>:    test   %rcx,%rcx
   0x000000000000172e <+14>:    jle    0x19a3 <calGrey()+643>
   0x0000000000001734 <+20>:    mov    0x3a85(%rip),%rdi        # 0x51c0 <G>
   0x000000000000173b <+27>:    mov    0x3a3e(%rip),%rdx        # 0x5180
<dstdata>
   0x0000000000001742 <+34>:    mov    0x3a97(%rip),%rsi        # 0x51e0 <R>
   0x0000000000001749 <+41>:    lea    0x1f(%rdi),%rax
   0x000000000000174d <+45>:    sub    %rdx,%rax
   0x0000000000001750 <+48>:    cmp    $0x3e,%rax
   0x0000000000001754 <+52>:    lea    0x1f(%rsi),%r9
   0x0000000000001758 <+56>:    seta   %al
   0x000000000000175b <+59>:    sub    %rdx,%r9
   0x000000000000175e <+62>:    cmp    $0x3e,%r9
   0x0000000000001762 <+66>:    seta   %r9b
   0x0000000000001766 <+70>:    and    %r9d,%eax
   0x0000000000001769 <+73>:    lea    -0x1(%rcx),%r9
   0x000000000000176d <+77>:    cmp    $0x6,%r9
   0x0000000000001771 <+81>:    seta   %r9b
   0x0000000000001775 <+85>:    mov    0x3a24(%rip),%r8         # 0x51a0 <B>
   0x000000000000177c <+92>:    test   %r9b,%al
   0x000000000000177f <+95>:    je     0x19a8 <calGrey()+648>
   0x0000000000001785 <+101>:   lea    0x1f(%r8),%rax
   0x0000000000001789 <+105>:   sub    %rdx,%rax
```

```
0x000000000000178c <+108>:   cmp    $0x3e,%rax
0x0000000000001790 <+112>:   jbe    0x19a8 <calGrey()+648>
0x0000000000001796 <+118>:   mov    %rcx,%r9
0x0000000000001799 <+121>:   shr    $0x3,%r9
0x000000000000179d <+125>:   vmovdqa 0x18db(%rip),%ymm6        # 0x3080
0x00000000000017a5 <+133>:   vmovdqa 0x18f3(%rip),%ymm5        # 0x30a0
0x00000000000017ad <+141>:   vmovdqa 0x190b(%rip),%ymm4        # 0x30c0
0x00000000000017b5 <+149>:   shl    $0x5,%r9
0x00000000000017b9 <+153>:   xor    %eax,%eax
0x00000000000017bb <+155>:   vpxor  %xmm3,%xmm3,%xmm3
0x00000000000017bf <+159>:   nop
0x00000000000017c0 <+160>:   vmovdqu (%r8,%rax,1),%ymm2
0x00000000000017c6 <+166>:   vpmulld (%rdi,%rax,1),%ymm5,%ymm1
0x00000000000017cc <+172>:   vpmulld (%rsi,%rax,1),%ymm6,%ymm0
0x00000000000017d2 <+178>:   vpaddd %ymm1,%ymm0,%ymm0
0x00000000000017d6 <+182>:   vpslld $0x3,%ymm2,%ymm1
0x00000000000017db <+187>:   vpsubd %ymm2,%ymm1,%ymm1
0x00000000000017df <+191>:   vpslld $0x2,%ymm1,%ymm1
0x00000000000017e4 <+196>:   vpaddd %ymm1,%ymm0,%ymm1
0x00000000000017e8 <+200>:   vpcmpgtd %ymm1,%ymm3,%ymm0
0x00000000000017ec <+204>:   vpand  %ymm4,%ymm0,%ymm0
0x00000000000017f0 <+208>:   vpaddd %ymm1,%ymm0,%ymm0
0x00000000000017f4 <+212>:   vpsrad $0x8,%ymm0,%ymm0
0x00000000000017f9 <+217>:   vmovdqu %ymm0,(%rdx,%rax,1)
0x00000000000017fe <+222>:   add    $0x20,%rax
0x0000000000001802 <+226>:   cmp    %r9,%rax
0x0000000000001805 <+229>:   jne    0x17c0 <calGrey()+160>
0x0000000000001807 <+231>:   mov    %rcx,%rax
0x000000000000180a <+234>:   and    $0xfffffffffffffff8,%rax
0x000000000000180e <+238>:   test   $0x7,%cl
0x0000000000001811 <+241>:   je     0x19a0 <calGrey()+640>
0x0000000000001817 <+247>:   movslq %eax,%r11
0x000000000000181a <+250>:   imul   $0x97,(%rdi,%r11,4),%r10d
0x0000000000001822 <+258>:   imul   $0x4d,(%rsi,%r11,4),%r9d
0x0000000000001827 <+263>:   add    %r10d,%r9d
0x000000000000182a <+266>:   imul   $0x1c,(%r8,%r11,4),%r10d
0x000000000000182f <+271>:   add    %r10d,%r9d
0x0000000000001832 <+274>:   lea    0xff(%r9),%r10d
0x0000000000001839 <+281>:   cmovs  %r10d,%r9d
0x000000000000183d <+285>:   sar    $0x8,%r9d
0x0000000000001841 <+289>:   mov    %r9d,(%rdx,%r11,4)
0x0000000000001845 <+293>:   lea    0x1(%rax),%r9d
0x0000000000001849 <+297>:   movslq %r9d,%r9
0x000000000000184c <+300>:   cmp    %r9,%rcx
0x000000000000184f <+303>:   jle    0x19a0 <calGrey()+640>
0x0000000000001855 <+309>:   imul   $0x97,(%rdi,%r9,4),%r11d
0x000000000000185d <+317>:   imul   $0x4d,(%rsi,%r9,4),%r10d
0x0000000000001862 <+322>:   add    %r11d,%r10d
0x0000000000001865 <+325>:   imul   $0x1c,(%r8,%r9,4),%r11d
0x000000000000186a <+330>:   add    %r11d,%r10d
0x000000000000186d <+333>:   lea    0xff(%r10),%r11d
0x0000000000001874 <+340>:   cmovs  %r11d,%r10d
0x0000000000001878 <+344>:   sar    $0x8,%r10d
0x000000000000187c <+348>:   mov    %r10d,(%rdx,%r9,4)
0x0000000000001880 <+352>:   lea    0x2(%rax),%r9d
0x0000000000001884 <+356>:   movslq %r9d,%r9
0x0000000000001887 <+359>:   cmp    %r9,%rcx
0x000000000000188a <+362>:   jle    0x19a0 <calGrey()+640>
```

```
0x0000000000001890 <+368>:   imul   $0x97,(%rdi,%r9,4),%r11d
0x0000000000001898 <+376>:   imul   $0x4d,(%rsi,%r9,4),%r10d
0x000000000000189d <+381>:   add    %r11d,%r10d
0x00000000000018a0 <+384>:   imul   $0x1c,(%r8,%r9,4),%r11d
0x00000000000018a5 <+389>:   add    %r11d,%r10d
0x00000000000018a8 <+392>:   lea    0xff(%r10),%r11d
0x00000000000018af <+399>:   cmovs  %r11d,%r10d
0x00000000000018b3 <+403>:   sar    $0x8,%r10d
0x00000000000018b7 <+407>:   mov    %r10d,(%rdx,%r9,4)
0x00000000000018bb <+411>:   lea    0x3(%rax),%r9d
0x00000000000018bf <+415>:   movslq %r9d,%r9
0x00000000000018c2 <+418>:   cmp    %r9,%rcx
0x00000000000018c5 <+421>:   jle    0x19a0 <calGrey()+640>
0x00000000000018cb <+427>:   imul   $0x97,(%rdi,%r9,4),%r11d
0x00000000000018d3 <+435>:   imul   $0x4d,(%rsi,%r9,4),%r10d
0x00000000000018d8 <+440>:   add    %r11d,%r10d
0x00000000000018db <+443>:   imul   $0x1c,(%r8,%r9,4),%r11d
0x00000000000018e0 <+448>:   add    %r11d,%r10d
0x00000000000018e3 <+451>:   lea    0xff(%r10),%r11d
0x00000000000018ea <+458>:   cmovs  %r11d,%r10d
0x00000000000018ee <+462>:   sar    $0x8,%r10d
0x00000000000018f2 <+466>:   mov    %r10d,(%rdx,%r9,4)
0x00000000000018f6 <+470>:   lea    0x4(%rax),%r9d
0x00000000000018fa <+474>:   movslq %r9d,%r9
0x00000000000018fd <+477>:   cmp    %r9,%rcx
0x0000000000001900 <+480>:   jle    0x19a0 <calGrey()+640>
0x0000000000001906 <+486>:   imul   $0x97,(%rdi,%r9,4),%r11d
0x000000000000190e <+494>:   imul   $0x4d,(%rsi,%r9,4),%r10d
0x0000000000001913 <+499>:   add    %r11d,%r10d
0x0000000000001916 <+502>:   imul   $0x1c,(%r8,%r9,4),%r11d
0x000000000000191b <+507>:   add    %r11d,%r10d
0x000000000000191e <+510>:   lea    0xff(%r10),%r11d
0x0000000000001925 <+517>:   cmovs  %r11d,%r10d
0x0000000000001929 <+521>:   sar    $0x8,%r10d
0x000000000000192d <+525>:   mov    %r10d,(%rdx,%r9,4)
0x0000000000001931 <+529>:   lea    0x5(%rax),%r9d
0x0000000000001935 <+533>:   movslq %r9d,%r9
0x0000000000001938 <+536>:   cmp    %r9,%rcx
0x000000000000193b <+539>:   jle    0x19a0 <calGrey()+640>
0x000000000000193d <+541>:   imul   $0x97,(%rdi,%r9,4),%r11d
0x0000000000001945 <+549>:   imul   $0x4d,(%rsi,%r9,4),%r10d
0x000000000000194a <+554>:   add    %r11d,%r10d
0x000000000000194d <+557>:   imul   $0x1c,(%r8,%r9,4),%r11d
0x0000000000001952 <+562>:   add    %r11d,%r10d
0x0000000000001955 <+565>:   lea    0xff(%r10),%r11d
0x000000000000195c <+572>:   cmovs  %r11d,%r10d
0x0000000000001960 <+576>:   add    $0x6,%eax
0x0000000000001963 <+579>:   sar    $0x8,%r10d
0x0000000000001967 <+583>:   cltq
0x0000000000001969 <+585>:   mov    %r10d,(%rdx,%r9,4)
0x000000000000196d <+589>:   cmp    %rax,%rcx
0x0000000000001970 <+592>:   jle    0x19a0 <calGrey()+640>
0x0000000000001972 <+594>:   imul   $0x4d,(%rsi,%rax,4),%ecx
0x0000000000001976 <+598>:   imul   $0x97,(%rdi,%rax,4),%esi
0x000000000000197d <+605>:   add    %esi,%ecx
0x000000000000197f <+607>:   imul   $0x1c,(%r8,%rax,4),%esi
0x0000000000001984 <+612>:   add    %esi,%ecx
0x0000000000001986 <+614>:   lea    0xff(%rcx),%esi
```

```
   0x000000000000198c <+620>:   cmovs  %esi,%ecx
   0x000000000000198f <+623>:   sar    $0x8,%ecx
   0x0000000000001992 <+626>:   mov    %ecx,(%rdx,%rax,4)
   0x0000000000001995 <+629>:   vzeroupper
   0x0000000000001998 <+632>:   retq
   0x0000000000001999 <+633>:   nopl   0x0(%rax)
   0x00000000000019a0 <+640>:   vzeroupper
   0x00000000000019a3 <+643>:   retq
   0x00000000000019a4 <+644>:   nopl   0x0(%rax)
   0x00000000000019a8 <+648>:   lea    0x0(,%rcx,4),%r9
   0x00000000000019b0 <+656>:   xor    %ecx,%ecx
   0x00000000000019b2 <+658>:   nopw   0x0(%rax,%rax,1)
   0x00000000000019b8 <+664>:   imul   $0x97,(%rdi,%rcx,1),%r10d
   0x00000000000019c0 <+672>:   imul   $0x4d,(%rsi,%rcx,1),%eax
   0x00000000000019c4 <+676>:   add    %r10d,%eax
   0x00000000000019c7 <+679>:   imul   $0x1c,(%r8,%rcx,1),%r10d
   0x00000000000019cc <+684>:   add    %r10d,%eax
   0x00000000000019cf <+687>:   lea    0xff(%rax),%r10d
   0x00000000000019d6 <+694>:   cmovs  %r10d,%eax
   0x00000000000019da <+698>:   sar    $0x8,%eax
   0x00000000000019dd <+701>:   mov    %eax,(%rdx,%rcx,1)
   0x00000000000019e0 <+704>:   add    $0x4,%rcx
   0x00000000000019e4 <+708>:   cmp    %r9,%rcx
   0x00000000000019e7 <+711>:   jne    0x19b8 <calGrey()+664>
   0x00000000000019e9 <+713>:   retq
```

可以看到，出现了SIMD指令，并且从使用了YMM寄存器可以看出编译器默认使用的SIMD指令集是AVX。分析汇编指令可以看出，函数的执行逻辑变成了：

1. 0x20之前：函数首先判断控制循环的`length`是否为0，如果是，则直接结束函数。否则进入循环。
2. 0x20-0x159：获取四个数组的首地址，并且将常数77、151、28放进ymm6、ymm5和ymm4。为进入SIMD的循环做一些准备。
3. 0x160-0x229：全是SIMD指令的循环，其实占整个函数的大部分循环。同时，分析这里用到的算术指令，可以看出函数并不是简单地执行了三次乘法、两次加法和一次右移（除法），而是通过加法、乘法、移位、与等一系列运算来获得计算结果。
4. 0x231-0x643：执行完大部分循环后，因为长度`length`不一定是8的整数（一个256位的YMM寄存器能存放8个32位的整型元素），会有剩余的元素需要单独计算。事实上，这一部分代码是重复的，每一段都是在计算`dstdata[i] = ((R[i] * 77 + G[i] * 151 + B[i] * 28 ) / 256);`。编译器把一个次数肯定小于8的循环展开了。每计算完一段，判断是否已经把所有元素计算完了，计算完则直接退出函数。退出函数之前的`vzeroupper`指令是使用YMM寄存器的要求。
5. 0x644-0x713：这一部分应该是`length`小于8的时候执行的指令。因为`length`小于8，元素太少不足以使用SIMD指令进行循环，因此进入该循环逐个元素进行计算。

intrinsic指令向量化

指定优化等级为三级`g++ -g image.cpp bmp.cpp -o image -msse4.1 -mfpmath=sse -march=native -O3 -Wall`，同时在程序中使用带有intrinsic指令的`calGrey_sse()`。可以看到执行时间约为0.0018秒。

```
cheung@cheung:~/Desktop/code/cpp$ g++ -g image.cpp bmp.cpp -o image -msse4.1 -mfpmath=sse -march=native -O3 -Wall
bmp.cpp: In function 'void loadBMPheader(FILE*, Header&)':
bmp.cpp:6:10: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unu
sed_result [-Wunused-result]
    6 |     fread(&header, 1, 54, file);
      |     ~~~~~^~~~~~~~~~~~~~~~~~~~~~
bmp.cpp: In function 'void loadBMPdata(FILE*, long int, int*, int*, int*)':
bmp.cpp:32:14: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_un
used_result [-Wunused-result]
   32 |         fread(rgb, 1, 3, file);
      |         ~~~~~^~~~~~~~~~~~~~~~~
cheung@cheung:~/Desktop/code/cpp$ ./image
Using intrinsic
Time for calculation: 0.002281
cheung@cheung:~/Desktop/code/cpp$ ./image
Using intrinsic
Time for calculation: 0.001714
cheung@cheung:~/Desktop/code/cpp$ ./image
Using intrinsic
Time for calculation: 0.002273
cheung@cheung:~/Desktop/code/cpp$
```

查看函数`calGrey_sse()`的汇编代码如下：

```
   0x00000000000019c0 <+0>:     endbr64
   0x00000000000019c4 <+4>:     push   %rbp
   0x00000000000019c5 <+5>:     mov    $0xf,%edx
   0x00000000000019ca <+10>:    lea    0x1633(%rip),%rsi        # 0x3004
   0x00000000000019d1 <+17>:    mov    %rsp,%rbp
   0x00000000000019d4 <+20>:    push   %r12
   0x00000000000019d6 <+22>:    lea    0x3663(%rip),%rdi        # 0x5040
<_ZSt4cout@@GLIBCXX_3.4>
   0x00000000000019dd <+29>:    push   %rbx
   0x00000000000019de <+30>:    and    $0xfffffffffffffff0,%rsp
   0x00000000000019e2 <+34>:    sub    $0x40,%rsp
   0x00000000000019e6 <+38>:    mov    %fs:0x28,%rax
   0x00000000000019ef <+47>:    mov    %rax,0x38(%rsp)
   0x00000000000019f4 <+52>:    xor    %eax,%eax
   0x00000000000019f6 <+54>:    callq  0x11e0
<_ZSt16__ostream_insertIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_PKS3_l@p
lt>
   0x00000000000019fb <+59>:    mov    0x363e(%rip),%rax        # 0x5040
<_ZSt4cout@@GLIBCXX_3.4>
   0x0000000000001a02 <+66>:    lea    0x3637(%rip),%rdx        # 0x5040
<_ZSt4cout@@GLIBCXX_3.4>
   0x0000000000001a09 <+73>:    mov    -0x18(%rax),%rax
   0x0000000000001a0d <+77>:    mov    0xf0(%rdx,%rax,1),%r12
   0x0000000000001a15 <+85>:    test   %r12,%r12
   0x0000000000001a18 <+88>:    je     0x1e6f <calGrey_sse()+1199>
   0x0000000000001a1e <+94>:    cmpb   $0x0,0x38(%r12)
   0x0000000000001a24 <+100>:   je     0x1df8 <calGrey_sse()+1080>
   0x0000000000001a2a <+106>:   movsbl 0x43(%r12),%esi
   0x0000000000001a30 <+112>:   lea    0x3609(%rip),%rdi        # 0x5040
<_ZSt4cout@@GLIBCXX_3.4>
   0x0000000000001a37 <+119>:   callq  0x1170 <_ZNSo3putEc@plt>
   0x0000000000001a3c <+124>:   mov    %rax,%rdi
   0x0000000000001a3f <+127>:   callq  0x11a0 <_ZNSo5flushEv@plt>
   0x0000000000001a44 <+132>:   movabs $0x4d0000004d,%rax
   0x0000000000001a4e <+142>:   mov    %rax,0x20(%rsp)
   0x0000000000001a53 <+147>:   xor    %edx,%edx
   0x0000000000001a55 <+149>:   mov    0x3784(%rip),%r11        # 0x51e0 <R>
   0x0000000000001a5c <+156>:   mov    0x20(%rsp),%rax
   0x0000000000001a61 <+161>:   mov    0x3758(%rip),%r10        # 0x51c0 <G>
   0x0000000000001a68 <+168>:   mov    %rax,0x28(%rsp)
   0x0000000000001a6d <+173>:   movabs $0x9800000098,%rax
   0x0000000000001a77 <+183>:   mov    %rax,0x10(%rsp)
   0x0000000000001a7c <+188>:   mov    0x371d(%rip),%r9         # 0x51a0 <B>
   0x0000000000001a83 <+195>:   mov    0x10(%rsp),%rax
```

```
   0x0000000000001a88 <+200>:   mov    %rax,0x18(%rsp)
   0x0000000000001a8d <+205>:   movabs $0x1c0000001c,%rax
   0x0000000000001a97 <+215>:   mov    %rax,(%rsp)
   0x0000000000001a9b <+219>:   mov    (%rsp),%rax
   0x0000000000001a9f <+223>:   mov    %rax,0x8(%rsp)
   0x0000000000001aa4 <+228>:   mov    0x36b5(%rip),%rax        # 0x5160
<length>
   0x0000000000001aab <+235>:   mov    %rax,%rbx
   0x0000000000001aae <+238>:   shr    $0x2,%rbx
   0x0000000000001ab2 <+242>:   div    %rbx
   0x0000000000001ab5 <+245>:   mov    0x36c4(%rip),%rax        # 0x5180
<dstdata>
   0x0000000000001abc <+252>:   test   %rbx,%rbx
   0x0000000000001abf <+255>:   je     0x1b88 <calGrey_sse()+456>
   0x0000000000001ac5 <+261>:   vmovdqa (%rsp),%xmm4
   0x0000000000001aca <+266>:   vmovdqa 0x10(%rsp),%xmm3
   0x0000000000001ad0 <+272>:   vmovdqa 0x20(%rsp),%xmm2
   0x0000000000001ad6 <+278>:   mov    %rax,%rcx
   0x0000000000001ad9 <+281>:   xor    %edi,%edi
   0x0000000000001adb <+283>:   xor    %r8d,%r8d
   0x0000000000001ade <+286>:   xchg   %ax,%ax
   0x0000000000001ae0 <+288>:   vpmulld (%r11,%rdi,1),%xmm2,%xmm1
   0x0000000000001ae6 <+294>:   vpmulld (%r10,%rdi,1),%xmm3,%xmm0
   0x0000000000001aec <+300>:   vpaddd %xmm1,%xmm0,%xmm0
   0x0000000000001af0 <+304>:   vpmulld (%r9,%rdi,1),%xmm4,%xmm1
   0x0000000000001af6 <+310>:   vpaddd %xmm1,%xmm0,%xmm0
   0x0000000000001afa <+314>:   vmovd  %xmm0,%r12d
   0x0000000000001aff <+319>:   vmovd  %xmm0,%esi
   0x0000000000001b03 <+323>:   test   %r12d,%r12d
   0x0000000000001b06 <+326>:   lea    0xff(%rsi),%esi
   0x0000000000001b0c <+332>:   vmovaps %xmm0,(%rcx)
   0x0000000000001b10 <+336>:   cmovns %r12d,%esi
   0x0000000000001b14 <+340>:   mov    0x4(%rcx),%r12d
   0x0000000000001b18 <+344>:   sar    $0x8,%esi
   0x0000000000001b1b <+347>:   mov    %esi,(%rcx)
   0x0000000000001b1d <+349>:   test   %r12d,%r12d
   0x0000000000001b20 <+352>:   lea    0xff(%r12),%esi
   0x0000000000001b28 <+360>:   cmovns %r12d,%esi
   0x0000000000001b2c <+364>:   mov    0x8(%rcx),%r12d
   0x0000000000001b30 <+368>:   sar    $0x8,%esi
   0x0000000000001b33 <+371>:   mov    %esi,0x4(%rcx)
   0x0000000000001b36 <+374>:   test   %r12d,%r12d
   0x0000000000001b39 <+377>:   lea    0xff(%r12),%esi
   0x0000000000001b41 <+385>:   cmovns %r12d,%esi
   0x0000000000001b45 <+389>:   mov    0xc(%rcx),%r12d
   0x0000000000001b49 <+393>:   sar    $0x8,%esi
   0x0000000000001b4c <+396>:   mov    %esi,0x8(%rcx)
   0x0000000000001b4f <+399>:   test   %r12d,%r12d
   0x0000000000001b52 <+402>:   lea    0xff(%r12),%esi
   0x0000000000001b5a <+410>:   cmovns %r12d,%esi
   0x0000000000001b5e <+414>:   inc    %r8
   0x0000000000001b61 <+417>:   sar    $0x8,%esi
   0x0000000000001b64 <+420>:   mov    %esi,0xc(%rcx)
   0x0000000000001b67 <+423>:   add    $0x10,%rdi
   0x0000000000001b6b <+427>:   add    $0x10,%rcx
   0x0000000000001b6f <+431>:   cmp    %r8,%rbx
   0x0000000000001b72 <+434>:   jne    0x1ae0 <calGrey_sse()+288>
   0x0000000000001b78 <+440>:   shl    $0x4,%rbx
```

```
0x0000000000001b7c <+444>:   add    %rbx,%r11
0x0000000000001b7f <+447>:   add    %rbx,%r10
0x0000000000001b82 <+450>:   add    %rbx,%r9
0x0000000000001b85 <+453>:   add    %rbx,%rax
0x0000000000001b88 <+456>:   test   %rdx,%rdx
0x0000000000001b8b <+459>:   je     0x1dce <calGrey_sse()+1038>
0x0000000000001b91 <+465>:   lea    0x1f(%rax),%rcx
0x0000000000001b95 <+469>:   mov    %rcx,%rsi
0x0000000000001b98 <+472>:   sub    %r10,%rsi
0x0000000000001b9b <+475>:   cmp    $0x3e,%rsi
0x0000000000001b9f <+479>:   mov    %rcx,%rdi
0x0000000000001ba2 <+482>:   seta   %sil
0x0000000000001ba6 <+486>:   sub    %r9,%rdi
0x0000000000001ba9 <+489>:   cmp    $0x3e,%rdi
0x0000000000001bad <+493>:   seta   %dil
0x0000000000001bb1 <+497>:   and    %edi,%esi
0x0000000000001bb3 <+499>:   lea    -0x1(%rdx),%rdi
0x0000000000001bb7 <+503>:   cmp    $0x6,%rdi
0x0000000000001bbb <+507>:   seta   %dil
0x0000000000001bbf <+511>:   test   %dil,%sil
0x0000000000001bc2 <+514>:   je     0x1e30 <calGrey_sse()+1136>
0x0000000000001bc8 <+520>:   sub    %r11,%rcx
0x0000000000001bcb <+523>:   cmp    $0x3e,%rcx
0x0000000000001bcf <+527>:   jbe    0x1e30 <calGrey_sse()+1136>
0x0000000000001bd5 <+533>:   mov    %rdx,%rsi
0x0000000000001bd8 <+536>:   shr    $0x3,%rsi
0x0000000000001bdc <+540>:   vmovdqa 0x14fc(%rip),%ymm4        # 0x30e0
0x0000000000001be4 <+548>:   vmovdqa 0x1514(%rip),%ymm3        # 0x3100
0x0000000000001bec <+556>:   shl    $0x5,%rsi
0x0000000000001bf0 <+560>:   xor    %ecx,%ecx
0x0000000000001bf2 <+562>:   vpxor  %xmm2,%xmm2,%xmm2
0x0000000000001bf6 <+566>:   nopw   %cs:0x0(%rax,%rax,1)
0x0000000000001c00 <+576>:   vmovdqu (%r10,%rcx,1),%ymm5
0x0000000000001c06 <+582>:   vpmulld (%r11,%rcx,1),%ymm4,%ymm1
0x0000000000001c0c <+588>:   vpslld $0x2,%ymm5,%ymm0
0x0000000000001c11 <+593>:   vpaddd %ymm5,%ymm0,%ymm0
0x0000000000001c15 <+597>:   vpslld $0x2,%ymm0,%ymm0
0x0000000000001c1a <+602>:   vpsubd %ymm5,%ymm0,%ymm0
0x0000000000001c1e <+606>:   vmovdqu (%r9,%rcx,1),%ymm5
0x0000000000001c24 <+612>:   vpslld $0x3,%ymm0,%ymm0
0x0000000000001c29 <+617>:   vpaddd %ymm0,%ymm1,%ymm0
0x0000000000001c2d <+621>:   vpslld $0x3,%ymm5,%ymm1
0x0000000000001c32 <+626>:   vpsubd %ymm5,%ymm1,%ymm1
0x0000000000001c36 <+630>:   vpslld $0x2,%ymm1,%ymm1
0x0000000000001c3b <+635>:   vpaddd %ymm1,%ymm0,%ymm1
0x0000000000001c3f <+639>:   vpcmpgtd %ymm1,%ymm2,%ymm0
0x0000000000001c43 <+643>:   vpand  %ymm3,%ymm0,%ymm0
0x0000000000001c47 <+647>:   vpaddd %ymm1,%ymm0,%ymm0
0x0000000000001c4b <+651>:   vpsrad $0x8,%ymm0,%ymm0
0x0000000000001c50 <+656>:   vmovdqu %ymm0,(%rax,%rcx,1)
0x0000000000001c55 <+661>:   add    $0x20,%rcx
0x0000000000001c59 <+665>:   cmp    %rcx,%rsi
0x0000000000001c5c <+668>:   jne    0x1c00 <calGrey_sse()+576>
0x0000000000001c5e <+670>:   mov    %rdx,%rcx
0x0000000000001c61 <+673>:   and    $0xfffffffffffffff8,%rcx
0x0000000000001c65 <+677>:   cmp    %rcx,%rdx
0x0000000000001c68 <+680>:   je     0x1df0 <calGrey_sse()+1072>
0x0000000000001c6e <+686>:   imul   $0x98,(%r10,%rcx,4),%edi
```

```
0x0000000000001c76 <+694>:   imul   $0x4d,(%r11,%rcx,4),%esi
0x0000000000001c7b <+699>:   add    %edi,%esi
0x0000000000001c7d <+701>:   imul   $0x1c,(%r9,%rcx,4),%edi
0x0000000000001c82 <+706>:   add    %edi,%esi
0x0000000000001c84 <+708>:   lea    0xff(%rsi),%edi
0x0000000000001c8a <+714>:   cmovs  %edi,%esi
0x0000000000001c8d <+717>:   lea    0x1(%rcx),%rdi
0x0000000000001c91 <+721>:   sar    $0x8,%esi
0x0000000000001c94 <+724>:   mov    %esi,(%rax,%rcx,4)
0x0000000000001c97 <+727>:   cmp    %rdi,%rdx
0x0000000000001c9a <+730>:   jbe    0x1df0 <calGrey_sse()+1072>
0x0000000000001ca0 <+736>:   imul   $0x98,(%r10,%rdi,4),%r8d
0x0000000000001ca8 <+744>:   imul   $0x4d,(%r11,%rdi,4),%esi
0x0000000000001cad <+749>:   add    %r8d,%esi
0x0000000000001cb0 <+752>:   imul   $0x1c,(%r9,%rdi,4),%r8d
0x0000000000001cb5 <+757>:   add    %r8d,%esi
0x0000000000001cb8 <+760>:   lea    0xff(%rsi),%r8d
0x0000000000001cbf <+767>:   cmovs  %r8d,%esi
0x0000000000001cc3 <+771>:   sar    $0x8,%esi
0x0000000000001cc6 <+774>:   mov    %esi,(%rax,%rdi,4)
0x0000000000001cc9 <+777>:   lea    0x2(%rcx),%rdi
0x0000000000001ccd <+781>:   cmp    %rdi,%rdx
0x0000000000001cd0 <+784>:   jbe    0x1df0 <calGrey_sse()+1072>
0x0000000000001cd6 <+790>:   imul   $0x98,(%r10,%rdi,4),%r8d
0x0000000000001cde <+798>:   imul   $0x4d,(%r11,%rdi,4),%esi
0x0000000000001ce3 <+803>:   add    %r8d,%esi
0x0000000000001ce6 <+806>:   imul   $0x1c,(%r9,%rdi,4),%r8d
0x0000000000001ceb <+811>:   add    %r8d,%esi
0x0000000000001cee <+814>:   lea    0xff(%rsi),%r8d
0x0000000000001cf5 <+821>:   cmovs  %r8d,%esi
0x0000000000001cf9 <+825>:   sar    $0x8,%esi
0x0000000000001cfc <+828>:   mov    %esi,(%rax,%rdi,4)
0x0000000000001cff <+831>:   lea    0x3(%rcx),%rdi
0x0000000000001d03 <+835>:   cmp    %rdi,%rdx
0x0000000000001d06 <+838>:   jbe    0x1df0 <calGrey_sse()+1072>
0x0000000000001d0c <+844>:   imul   $0x98,(%r10,%rdi,4),%r8d
0x0000000000001d14 <+852>:   imul   $0x4d,(%r11,%rdi,4),%esi
0x0000000000001d19 <+857>:   add    %r8d,%esi
0x0000000000001d1c <+860>:   imul   $0x1c,(%r9,%rdi,4),%r8d
0x0000000000001d21 <+865>:   add    %r8d,%esi
0x0000000000001d24 <+868>:   lea    0xff(%rsi),%r8d
0x0000000000001d2b <+875>:   cmovs  %r8d,%esi
0x0000000000001d2f <+879>:   sar    $0x8,%esi
0x0000000000001d32 <+882>:   mov    %esi,(%rax,%rdi,4)
0x0000000000001d35 <+885>:   lea    0x4(%rcx),%rdi
0x0000000000001d39 <+889>:   cmp    %rdi,%rdx
0x0000000000001d3c <+892>:   jbe    0x1df0 <calGrey_sse()+1072>
0x0000000000001d42 <+898>:   imul   $0x98,(%r10,%rdi,4),%r8d
0x0000000000001d4a <+906>:   imul   $0x4d,(%r11,%rdi,4),%esi
0x0000000000001d4f <+911>:   add    %r8d,%esi
0x0000000000001d52 <+914>:   imul   $0x1c,(%r9,%rdi,4),%r8d
0x0000000000001d57 <+919>:   add    %r8d,%esi
0x0000000000001d5a <+922>:   lea    0xff(%rsi),%r8d
0x0000000000001d61 <+929>:   cmovs  %r8d,%esi
0x0000000000001d65 <+933>:   sar    $0x8,%esi
0x0000000000001d68 <+936>:   mov    %esi,(%rax,%rdi,4)
0x0000000000001d6b <+939>:   lea    0x5(%rcx),%rdi
0x0000000000001d6f <+943>:   cmp    %rdi,%rdx
```

```
   0x0000000000001d72 <+946>:   jbe    0x1df0 <calGrey_sse()+1072>
   0x0000000000001d74 <+948>:   imul   $0x98,(%r10,%rdi,4),%r8d
   0x0000000000001d7c <+956>:   imul   $0x4d,(%r11,%rdi,4),%esi
   0x0000000000001d81 <+961>:   add    %r8d,%esi
   0x0000000000001d84 <+964>:   imul   $0x1c,(%r9,%rdi,4),%r8d
   0x0000000000001d89 <+969>:   add    %r8d,%esi
   0x0000000000001d8c <+972>:   lea    0xff(%rsi),%r8d
   0x0000000000001d93 <+979>:   cmovs  %r8d,%esi
   0x0000000000001d97 <+983>:   add    $0x6,%rcx
   0x0000000000001d9b <+987>:   sar    $0x8,%esi
   0x0000000000001d9e <+990>:   mov    %esi,(%rax,%rdi,4)
   0x0000000000001da1 <+993>:   cmp    %rcx,%rdx
   0x0000000000001da4 <+996>:   jbe    0x1df0 <calGrey_sse()+1072>
   0x0000000000001da6 <+998>:   imul   $0x4d,(%r11,%rcx,4),%esi
   0x0000000000001dab <+1003>:  imul   $0x98,(%r10,%rcx,4),%edx
   0x0000000000001db3 <+1011>:  add    %esi,%edx
   0x0000000000001db5 <+1013>:  imul   $0x1c,(%r9,%rcx,4),%esi
   0x0000000000001dba <+1018>:  add    %esi,%edx
   0x0000000000001dbc <+1020>:  lea    0xff(%rdx),%esi
   0x0000000000001dc2 <+1026>:  cmovs  %esi,%edx
   0x0000000000001dc5 <+1029>:  sar    $0x8,%edx
   0x0000000000001dc8 <+1032>:  mov    %edx,(%rax,%rcx,4)
   0x0000000000001dcb <+1035>:  vzeroupper
   0x0000000000001dce <+1038>:  mov    0x38(%rsp),%rax
   0x0000000000001dd3 <+1043>:  xor    %fs:0x28,%rax
   0x0000000000001ddc <+1052>:  jne    0x1e6a <calGrey_sse()+1194>
   0x0000000000001de2 <+1058>:  lea    -0x10(%rbp),%rsp
   0x0000000000001de6 <+1062>:  pop    %rbx
   0x0000000000001de7 <+1063>:  pop    %r12
   0x0000000000001de9 <+1065>:  pop    %rbp
   0x0000000000001dea <+1066>:  retq
   0x0000000000001deb <+1067>:  nopl   0x0(%rax,%rax,1)
   0x0000000000001df0 <+1072>:  vzeroupper
   0x0000000000001df3 <+1075>:  jmp    0x1dce <calGrey_sse()+1038>
   0x0000000000001df5 <+1077>:  nopl   (%rax)
   0x0000000000001df8 <+1080>:  mov    %r12,%rdi
   0x0000000000001dfb <+1083>:  callq  0x11f0
<_ZNKSt5ctypeIcE13_M_widen_initEv@plt>
   0x0000000000001e00 <+1088>:  mov    (%r12),%rax
   0x0000000000001e04 <+1092>:  lea    0x75(%rip),%rdx        # 0x1e80
<std::ctype<char>::do_widen(char) const>
   0x0000000000001e0b <+1099>:  mov    0x30(%rax),%rax
   0x0000000000001e0f <+1103>:  mov    $0xa,%esi
   0x0000000000001e14 <+1108>:  cmp    %rdx,%rax
   0x0000000000001e17 <+1111>:  je     0x1a30 <calGrey_sse()+112>
   0x0000000000001e1d <+1117>:  mov    %r12,%rdi
   0x0000000000001e20 <+1120>:  callq  *%rax
   0x0000000000001e22 <+1122>:  movsbl %al,%esi
   0x0000000000001e25 <+1125>:  jmpq   0x1a30 <calGrey_sse()+112>
   0x0000000000001e2a <+1130>:  nopw   0x0(%rax,%rax,1)
   0x0000000000001e30 <+1136>:  xor    %esi,%esi
   0x0000000000001e32 <+1138>:  nopw   0x0(%rax,%rax,1)
   0x0000000000001e38 <+1144>:  imul   $0x4d,(%r11,%rsi,4),%edi
   0x0000000000001e3d <+1149>:  imul   $0x98,(%r10,%rsi,4),%ecx
   0x0000000000001e45 <+1157>:  add    %edi,%ecx
   0x0000000000001e47 <+1159>:  imul   $0x1c,(%r9,%rsi,4),%edi
   0x0000000000001e4c <+1164>:  add    %edi,%ecx
   0x0000000000001e4e <+1166>:  lea    0xff(%rcx),%edi
```

```
   0x0000000000001e54 <+1172>:  cmovs  %edi,%ecx
   0x0000000000001e57 <+1175>:  sar    $0x8,%ecx
   0x0000000000001e5a <+1178>:  mov    %ecx,(%rax,%rsi,4)
   0x0000000000001e5d <+1181>:  inc    %rsi
   0x0000000000001e60 <+1184>:  cmp    %rsi,%rdx
   0x0000000000001e63 <+1187>:  jne    0x1e38 <calGrey_sse()+1144>
   0x0000000000001e65 <+1189>:  jmpq   0x1dce <calGrey_sse()+1038>
   0x0000000000001e6a <+1194>:  callq  0x11d0 <__stack_chk_fail@plt>
   0x0000000000001e6f <+1199>:  callq  0x1210 <_ZSt16__throw_bad_castv@plt>
```

可以看到，同样出现了SIMD指令，并且使用了XMM寄存器可以看出编译器确实使用了代码指定的SSE指令集。但是比较奇怪的是，同时也出现了使用YMM寄存器的AVX指令。由于这其中出现了几个系统调用，我无法得知系统调用的具体用途，分析汇编指令可以看出，我对函数执行逻辑的猜测如下：

1. 0x286之前：函数对是否有必要进入使用SIMD的循环进行检查，如果是，则继续执行使用SIMD的循环，否则逐个元素进行计算或者直接退出。同时，在这一部分还对硬件是否支持SSE进行了检查，如果支持SSE，则进入由intrinsic函数指定的SSE循环，否则进入由AVX指令构成的SIMD循环。我猜测这么做的原因是x86默认支持AVX但不一定还支持相对更古老的SSE。
2. 0x288-0x434：由intrinsic函数指定的SSE循环。可以看到，这里出现的SIMD指令（三个乘法和两个加法）和随后的四步除法有着非常直接的对应关系。验证了intrinsic函数的作用。
3. 0x576-0x1066：全是AVX指令的循环。对比普通循环向量化的汇编指令来看，可以看出这里的计算逻辑是相似的，不是直接的乘法加法而是包括了算术运算和位运算等一系列运算。其后，就是处理剩余的元素，展开的七次循环。与上一部分是一样的。
4. 0x1067-0x1189：与检查对SSE指令集的支持有关，同时还包括了一个处理SSE循环处理剩余的元素的循环。其计算逻辑也是intrinsic指定的三个乘法、两个加法和一个移位（除法）。
5. 0x1194-0x1199：两个系统调用，应该是处理异常或执行问题的。

在这里，我吃惊地发现这样做的执行时间比编译器优化普通代码的执行时间还长，我认为这不太合理。我猜测是因为SSE没有整数除法的指令，串行执行除法的代码拖慢了运行速度。为了验证想法，我把除法运算注释掉之后又测试了一次。

intrinsic指令向量化（无串行除法）

指定优化等级为三级`g++ -g image.cpp bmp.cpp -o image -msse4.1 -mfpmath=sse -march=native -O3 -Wall`，同时在程序中使用带有intrinsic指令的`calGrey_sse()`，注释掉除法运算。可以看到执行时间约为0.0017秒。

```
cheung@cheung:~/Desktop/code/cpp$ g++ -g image.cpp bmp.cpp -o image -msse4.1 -mfpmath=sse -march=native -O3 -Wall
bmp.cpp: In function 'void loadBMPheader(FILE*, Header&)':
bmp.cpp:6:10: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unu
sed_result [-Wunused-result]
    6 |     fread(&header, 1, 54, file);
      |     ~~~~~^~~~~~~~~~~~~~~~~~~~~~
bmp.cpp: In function 'void loadBMPdata(FILE*, long int, int*, int*, int*)':
bmp.cpp:32:14: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_un
used_result [-Wunused-result]
   32 |         fread(rgb, 1, 3, file);
      |         ~~~~~^~~~~~~~~~~~~~~~~
cheung@cheung:~/Desktop/code/cpp$ ./image
Using intrinsic
Time for calculation: 0.002147
cheung@cheung:~/Desktop/code/cpp$ ./image
Using intrinsic
Time for calculation: 0.001791
cheung@cheung:~/Desktop/code/cpp$ ./image
Using intrinsic
Time for calculation: 0.001863
cheung@cheung:~/Desktop/code/cpp$ ./image
Using intrinsic
Time for calculation: 0.001723
cheung@cheung:~/Desktop/code/cpp$
```

查看函数`calGrey_sse()`的汇编代码如下：

```
   0x00000000000019c0 <+0>:     endbr64
   0x00000000000019c4 <+4>:     push   %rbp
```

```
   0x00000000000019c5 <+5>:     mov    $0xf,%edx
   0x00000000000019ca <+10>:    lea    0x1633(%rip),%rsi        # 0x3004
   0x00000000000019d1 <+17>:    mov    %rsp,%rbp
   0x00000000000019d4 <+20>:    push   %r12
   0x00000000000019d6 <+22>:    lea    0x3663(%rip),%rdi        # 0x5040
<_ZSt4cout@@GLIBCXX_3.4>
   0x00000000000019dd <+29>:    and    $0xfffffffffffffff0,%rsp
   0x00000000000019e1 <+33>:    sub    $0x40,%rsp
   0x00000000000019e5 <+37>:    mov    %fs:0x28,%rax
   0x00000000000019ee <+46>:    mov    %rax,0x38(%rsp)
   0x00000000000019f3 <+51>:    xor    %eax,%eax
   0x00000000000019f5 <+53>:    callq  0x11e0
<_ZSt16__ostream_insertIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_PKS3_l@p
lt>
   0x00000000000019fa <+58>:    mov    0x363f(%rip),%rax        # 0x5040
<_ZSt4cout@@GLIBCXX_3.4>
   0x0000000000001a01 <+65>:    lea    0x3638(%rip),%rdx        # 0x5040
<_ZSt4cout@@GLIBCXX_3.4>
   0x0000000000001a08 <+72>:    mov    -0x18(%rax),%rax
   0x0000000000001a0c <+76>:    mov    0xf0(%rdx,%rax,1),%r12
   0x0000000000001a14 <+84>:    test   %r12,%r12
   0x0000000000001a17 <+87>:    je     0x1d63 <calGrey_sse()+931>
   0x0000000000001a1d <+93>:    cmpb   $0x0,0x38(%r12)
   0x0000000000001a23 <+99>:    je     0x1cf8 <calGrey_sse()+824>
   0x0000000000001a29 <+105>:   movsbl 0x43(%r12),%esi
   0x0000000000001a2f <+111>:   lea    0x360a(%rip),%rdi        # 0x5040
<_ZSt4cout@@GLIBCXX_3.4>
   0x0000000000001a36 <+118>:   callq  0x1170 <_ZNSo3putEc@plt>
   0x0000000000001a3b <+123>:   mov    %rax,%rdi
   0x0000000000001a3e <+126>:   callq  0x11a0 <_ZNSo5flushEv@plt>
   0x0000000000001a43 <+131>:   movabs $0x4d0000004d,%rax
   0x0000000000001a4d <+141>:   mov    %rax,0x20(%rsp)
   0x0000000000001a52 <+146>:   xor    %edx,%edx
   0x0000000000001a54 <+148>:   mov    0x3785(%rip),%r10        # 0x51e0 <R>
   0x0000000000001a5b <+155>:   mov    0x20(%rsp),%rax
   0x0000000000001a60 <+160>:   mov    0x3759(%rip),%r9         # 0x51c0 <G>
   0x0000000000001a67 <+167>:   mov    %rax,0x28(%rsp)
   0x0000000000001a6c <+172>:   movabs $0x9800000098,%rax
   0x0000000000001a76 <+182>:   mov    %rax,0x10(%rsp)
   0x0000000000001a7b <+187>:   mov    0x371e(%rip),%r8         # 0x51a0 <B>
   0x0000000000001a82 <+194>:   mov    0x36f7(%rip),%rdi        # 0x5180
<dstdata>
   0x0000000000001a89 <+201>:   mov    0x10(%rsp),%rax
   0x0000000000001a8e <+206>:   mov    %rax,0x18(%rsp)
   0x0000000000001a93 <+211>:   movabs $0x1c0000001c,%rax
   0x0000000000001a9d <+221>:   mov    %rax,(%rsp)
   0x0000000000001aa1 <+225>:   mov    (%rsp),%rax
   0x0000000000001aa5 <+229>:   mov    %rax,0x8(%rsp)
   0x0000000000001aaa <+234>:   mov    0x36af(%rip),%rax        # 0x5160
<length>
   0x0000000000001ab1 <+241>:   mov    %rax,%r11
   0x0000000000001ab4 <+244>:   shr    $0x2,%r11
   0x0000000000001ab8 <+248>:   div    %r11
   0x0000000000001abb <+251>:   test   %r11,%r11
   0x0000000000001abe <+254>:   je     0x1b13 <calGrey_sse()+339>
   0x0000000000001ac0 <+256>:   vmovdqa (%rsp),%xmm4
   0x0000000000001ac5 <+261>:   vmovdqa 0x10(%rsp),%xmm3
   0x0000000000001acb <+267>:   vmovdqa 0x20(%rsp),%xmm2
```

```
0x0000000000001ad1 <+273>:   xor    %ecx,%ecx
0x0000000000001ad3 <+275>:   xor    %esi,%esi
0x0000000000001ad5 <+277>:   nopl   (%rax)
0x0000000000001ad8 <+280>:   vpmulld (%r10,%rcx,1),%xmm2,%xmm1
0x0000000000001ade <+286>:   vpmulld (%r9,%rcx,1),%xmm3,%xmm0
0x0000000000001ae4 <+292>:   vpaddd %xmm1,%xmm0,%xmm0
0x0000000000001ae8 <+296>:   inc    %rsi
0x0000000000001aeb <+299>:   vpmulld (%r8,%rcx,1),%xmm4,%xmm1
0x0000000000001af1 <+305>:   vpaddd %xmm1,%xmm0,%xmm0
0x0000000000001af5 <+309>:   vmovaps %xmm0,(%rdi,%rcx,1)
0x0000000000001afa <+314>:   add    $0x10,%rcx
0x0000000000001afe <+318>:   cmp    %rsi,%r11
0x0000000000001b01 <+321>:   jne    0x1ad8 <calGrey_sse()+280>
0x0000000000001b03 <+323>:   shl    $0x4,%r11
0x0000000000001b07 <+327>:   add    %r11,%r10
0x0000000000001b0a <+330>:   add    %r11,%r9
0x0000000000001b0d <+333>:   add    %r11,%r8
0x0000000000001b10 <+336>:   add    %r11,%rdi
0x0000000000001b13 <+339>:   test   %rdx,%rdx
0x0000000000001b16 <+342>:   je     0x1cd7 <calGrey_sse()+791>
0x0000000000001b1c <+348>:   lea    0x1f(%rdi),%rax
0x0000000000001b20 <+352>:   mov    %rax,%rcx
0x0000000000001b23 <+355>:   sub    %r9,%rcx
0x0000000000001b26 <+358>:   cmp    $0x3e,%rcx
0x0000000000001b2a <+362>:   mov    %rax,%rsi
0x0000000000001b2d <+365>:   seta   %cl
0x0000000000001b30 <+368>:   sub    %r8,%rsi
0x0000000000001b33 <+371>:   cmp    $0x3e,%rsi
0x0000000000001b37 <+375>:   seta   %sil
0x0000000000001b3b <+379>:   and    %esi,%ecx
0x0000000000001b3d <+381>:   lea    -0x1(%rdx),%rsi
0x0000000000001b41 <+385>:   cmp    $0x6,%rsi
0x0000000000001b45 <+389>:   seta   %sil
0x0000000000001b49 <+393>:   test   %sil,%cl
0x0000000000001b4c <+396>:   je     0x1d30 <calGrey_sse()+880>
0x0000000000001b52 <+402>:   sub    %r10,%rax
0x0000000000001b55 <+405>:   cmp    $0x3e,%rax
0x0000000000001b59 <+409>:   jbe    0x1d30 <calGrey_sse()+880>
0x0000000000001b5f <+415>:   mov    %rdx,%rax
0x0000000000001b62 <+418>:   shr    $0x3,%rax
0x0000000000001b66 <+422>:   vmovdqa 0x1572(%rip),%ymm3        # 0x30e0
0x0000000000001b6e <+430>:   shl    $0x5,%rax
0x0000000000001b72 <+434>:   xor    %ecx,%ecx
0x0000000000001b74 <+436>:   nopl   0x0(%rax)
0x0000000000001b78 <+440>:   vmovdqu (%r9,%rcx,1),%ymm2
0x0000000000001b7e <+446>:   vpmulld (%r10,%rcx,1),%ymm3,%ymm1
0x0000000000001b84 <+452>:   vpslld $0x2,%ymm2,%ymm0
0x0000000000001b89 <+457>:   vpaddd %ymm2,%ymm0,%ymm0
0x0000000000001b8d <+461>:   vpslld $0x2,%ymm0,%ymm0
0x0000000000001b92 <+466>:   vpsubd %ymm2,%ymm0,%ymm0
0x0000000000001b96 <+470>:   vmovdqu (%r8,%rcx,1),%ymm2
0x0000000000001b9c <+476>:   vpslld $0x3,%ymm0,%ymm0
0x0000000000001ba1 <+481>:   vpaddd %ymm0,%ymm1,%ymm0
0x0000000000001ba5 <+485>:   vpslld $0x3,%ymm2,%ymm1
0x0000000000001baa <+490>:   vpsubd %ymm2,%ymm1,%ymm1
0x0000000000001bae <+494>:   vpslld $0x2,%ymm1,%ymm1
0x0000000000001bb3 <+499>:   vpaddd %ymm1,%ymm0,%ymm0
0x0000000000001bb7 <+503>:   vmovdqu %ymm0,(%rdi,%rcx,1)
```

```
0x0000000000001bbc <+508>:    add    $0x20,%rcx
0x0000000000001bc0 <+512>:    cmp    %rax,%rcx
0x0000000000001bc3 <+515>:    jne    0x1b78 <calGrey_sse()+440>
0x0000000000001bc5 <+517>:    mov    %rdx,%rax
0x0000000000001bc8 <+520>:    and    $0xfffffffffffffff8,%rax
0x0000000000001bcc <+524>:    cmp    %rax,%rdx
0x0000000000001bcf <+527>:    je     0x1cf0 <calGrey_sse()+816>
0x0000000000001bd5 <+533>:    imul   $0x98,(%r9,%rax,4),%esi
0x0000000000001bdd <+541>:    imul   $0x4d,(%r10,%rax,4),%ecx
0x0000000000001be2 <+546>:    add    %esi,%ecx
0x0000000000001be4 <+548>:    imul   $0x1c,(%r8,%rax,4),%esi
0x0000000000001be9 <+553>:    add    %esi,%ecx
0x0000000000001beb <+555>:    mov    %ecx,(%rdi,%rax,4)
0x0000000000001bee <+558>:    lea    0x1(%rax),%rcx
0x0000000000001bf2 <+562>:    cmp    %rcx,%rdx
0x0000000000001bf5 <+565>:    jbe    0x1cf0 <calGrey_sse()+816>
0x0000000000001bfb <+571>:    imul   $0x98,(%r9,%rcx,4),%r11d
0x0000000000001c03 <+579>:    imul   $0x4d,(%r10,%rcx,4),%esi
0x0000000000001c08 <+584>:    add    %r11d,%esi
0x0000000000001c0b <+587>:    imul   $0x1c,(%r8,%rcx,4),%r11d
0x0000000000001c10 <+592>:    add    %r11d,%esi
0x0000000000001c13 <+595>:    mov    %esi,(%rdi,%rcx,4)
0x0000000000001c16 <+598>:    lea    0x2(%rax),%rcx
0x0000000000001c1a <+602>:    cmp    %rcx,%rdx
0x0000000000001c1d <+605>:    jbe    0x1cf0 <calGrey_sse()+816>
0x0000000000001c23 <+611>:    imul   $0x98,(%r9,%rcx,4),%r11d
0x0000000000001c2b <+619>:    imul   $0x4d,(%r10,%rcx,4),%esi
0x0000000000001c30 <+624>:    add    %r11d,%esi
0x0000000000001c33 <+627>:    imul   $0x1c,(%r8,%rcx,4),%r11d
0x0000000000001c38 <+632>:    add    %r11d,%esi
0x0000000000001c3b <+635>:    mov    %esi,(%rdi,%rcx,4)
0x0000000000001c3e <+638>:    lea    0x3(%rax),%rcx
0x0000000000001c42 <+642>:    cmp    %rcx,%rdx
0x0000000000001c45 <+645>:    jbe    0x1cf0 <calGrey_sse()+816>
0x0000000000001c4b <+651>:    imul   $0x98,(%r9,%rcx,4),%r11d
0x0000000000001c53 <+659>:    imul   $0x4d,(%r10,%rcx,4),%esi
0x0000000000001c58 <+664>:    add    %r11d,%esi
0x0000000000001c5b <+667>:    imul   $0x1c,(%r8,%rcx,4),%r11d
0x0000000000001c60 <+672>:    add    %r11d,%esi
0x0000000000001c63 <+675>:    mov    %esi,(%rdi,%rcx,4)
0x0000000000001c66 <+678>:    lea    0x4(%rax),%rcx
0x0000000000001c6a <+682>:    cmp    %rcx,%rdx
0x0000000000001c6d <+685>:    jbe    0x1cf0 <calGrey_sse()+816>
0x0000000000001c73 <+691>:    imul   $0x98,(%r9,%rcx,4),%r11d
0x0000000000001c7b <+699>:    imul   $0x4d,(%r10,%rcx,4),%esi
0x0000000000001c80 <+704>:    add    %r11d,%esi
0x0000000000001c83 <+707>:    imul   $0x1c,(%r8,%rcx,4),%r11d
0x0000000000001c88 <+712>:    add    %r11d,%esi
0x0000000000001c8b <+715>:    mov    %esi,(%rdi,%rcx,4)
0x0000000000001c8e <+718>:    lea    0x5(%rax),%rcx
0x0000000000001c92 <+722>:    cmp    %rcx,%rdx
0x0000000000001c95 <+725>:    jbe    0x1cf0 <calGrey_sse()+816>
0x0000000000001c97 <+727>:    imul   $0x98,(%r9,%rcx,4),%r11d
0x0000000000001c9f <+735>:    imul   $0x4d,(%r10,%rcx,4),%esi
0x0000000000001ca4 <+740>:    add    $0x6,%rax
0x0000000000001ca8 <+744>:    add    %r11d,%esi
0x0000000000001cab <+747>:    imul   $0x1c,(%r8,%rcx,4),%r11d
0x0000000000001cb0 <+752>:    add    %r11d,%esi
```

```
   0x0000000000001cb3 <+755>:   mov    %esi,(%rdi,%rcx,4)
   0x0000000000001cb6 <+758>:   cmp    %rax,%rdx
   0x0000000000001cb9 <+761>:   jbe    0x1cf0 <calGrey_sse()+816>
   0x0000000000001cbb <+763>:   imul   $0x4d,(%r10,%rax,4),%ecx
   0x0000000000001cc0 <+768>:   imul   $0x98,(%r9,%rax,4),%edx
   0x0000000000001cc8 <+776>:   add    %ecx,%edx
   0x0000000000001cca <+778>:   imul   $0x1c,(%r8,%rax,4),%ecx
   0x0000000000001ccf <+783>:   add    %ecx,%edx
   0x0000000000001cd1 <+785>:   mov    %edx,(%rdi,%rax,4)
   0x0000000000001cd4 <+788>:   vzeroupper
   0x0000000000001cd7 <+791>:   mov    0x38(%rsp),%rax
   0x0000000000001cdc <+796>:   xor    %fs:0x28,%rax
   0x0000000000001ce5 <+805>:   jne    0x1d5e <calGrey_sse()+926>
   0x0000000000001ce7 <+807>:   mov    -0x8(%rbp),%r12
   0x0000000000001ceb <+811>:   leaveq
   0x0000000000001cec <+812>:   retq
   0x0000000000001ced <+813>:   nopl   (%rax)
   0x0000000000001cf0 <+816>:   vzeroupper
   0x0000000000001cf3 <+819>:   jmp    0x1cd7 <calGrey_sse()+791>
   0x0000000000001cf5 <+821>:   nopl   (%rax)
   0x0000000000001cf8 <+824>:   mov    %r12,%rdi
   0x0000000000001cfb <+827>:   callq  0x11f0
<_ZNKSt5ctypeIcE13_M_widen_initEv@plt>
   0x0000000000001d00 <+832>:   mov    (%r12),%rax
   0x0000000000001d04 <+836>:   lea    0x65(%rip),%rdx        # 0x1d70
<std::ctype<char>::do_widen(char) const>
   0x0000000000001d0b <+843>:   mov    0x30(%rax),%rax
   0x0000000000001d0f <+847>:   mov    $0xa,%esi
   0x0000000000001d14 <+852>:   cmp    %rdx,%rax
   0x0000000000001d17 <+855>:   je     0x1a2f <calGrey_sse()+111>
   0x0000000000001d1d <+861>:   mov    %r12,%rdi
   0x0000000000001d20 <+864>:   callq  *%rax
   0x0000000000001d22 <+866>:   movsbl %al,%esi
   0x0000000000001d25 <+869>:   jmpq   0x1a2f <calGrey_sse()+111>
   0x0000000000001d2a <+874>:   nopw   0x0(%rax,%rax,1)
   0x0000000000001d30 <+880>:   xor    %ecx,%ecx
   0x0000000000001d32 <+882>:   nopw   0x0(%rax,%rax,1)
   0x0000000000001d38 <+888>:   imul   $0x4d,(%r10,%rcx,4),%eax
   0x0000000000001d3d <+893>:   imul   $0x98,(%r9,%rcx,4),%esi
   0x0000000000001d45 <+901>:   add    %eax,%esi
   0x0000000000001d47 <+903>:   imul   $0x1c,(%r8,%rcx,4),%eax
   0x0000000000001d4c <+908>:   add    %eax,%esi
   0x0000000000001d4e <+910>:   mov    %esi,(%rdi,%rcx,4)
   0x0000000000001d51 <+913>:   inc    %rcx
   0x0000000000001d54 <+916>:   cmp    %rcx,%rdx
   0x0000000000001d57 <+919>:   jne    0x1d38 <calGrey_sse()+888>
   0x0000000000001d59 <+921>:   jmpq   0x1cd7 <calGrey_sse()+791>
   0x0000000000001d5e <+926>:   callq  0x11d0 <__stack_chk_fail@plt>
   0x0000000000001d63 <+931>:   callq  0x1210 <_ZSt16__throw_bad_castv@plt>
```

可以看到，这一部分的汇编指令与上一部分的基本无异。区别就在于没有了串行的除法。

这么做的执行时间已经很接近编译器优化普通函数的执行时间，但是还是比后者略长，我分析原因如下：

1. 需要检查硬件是否支持SSE
2. 使用128位的XMM寄存器，速度不如使用256位的YMM寄存器快
3. 灰度计算的公式较为复杂，我指定的计算方法不如编译器优化出来的计算方法快

实验体会

本次实验我尝试了多种方法来探索向量化指令的使用。这一实验过程让我对intrinsic指令的使用有了更深的体会，比如指令的命名规律、数组的地址对齐、元素的长度对intrinsic指令的使用都有何影响。同时，在图片处理这个场景下，验证了SIMD的加速作用。我还对编译器的参数设置有了更多的了解，以后在编程时会注意使用便于编译器进行向量化的代码风格。

经验教训

- `_mm_mul_epi32()`会存放64位的计算结果，在本实验中需要使用`_mm_mullo_epi32()`，这个函数会存放计算结果的低32位
- 因为一开始想避免数据类型的转换，我没有直接使用浮点数来计算。但是后来发现SSE其实不支持长度为一个字节的元素的并行计算，我还是需要将`unsigned char`转化为`int`。现在往回看，其实可以一开始就采用浮点数和浮点数公式来实验，而不是用整型的灰度公式

参考文献

【1】SSE指令集Intrinsic函数使用

【2】BMP格式详解

【3】跨平台使用Intrinsic函数范例3

【4】优化汇编例程（14）

【5】Intel® 64 and IA-32 Architectures Software Developer Manuals