作业 4

1 Exercise 1: 熟悉 SIMD intrinsics 函数

对应函数如下:

```
\\4个并行的单精度浮点数除法
__m128 _mm_div_ps (__m128 a, __m128 b)

\\16个并行求 8 位无符号整数的最大值
__m128i _mm_max_epu8 (__m128i a, __m128i b)

\\8个并行的 16 位带符号短整数的算术右移
__m128i _mm_sra_epi16 (__m128i a, __m128i count)
```

2 Exercise 2: 阅读 SIMD 代码

如下表，着色部分为 SIMD 指令。

```
main:
.LFB4883:
    .cfi_startproc
    sub rsp, 104
    .cfi_def_cfa_offset 112
    pxor    xmm7, xmm7
    movsd   xmm0, QWORD PTR .LC0[rip]
    mov esi, OFFSET FLAT:.LC6
    movsd   xmm1, QWORD PTR .LC1[rip]
    mov edi, 1
    movsd   QWORD PTR [rsp+16], xmm0
    pxor    xmm6, xmm6
    movsd   QWORD PTR [rsp+24], xmm1
    movsd   xmm5, QWORD PTR .LC3[rip]
    movsd   QWORD PTR [rsp+48], xmm6
    movupd  xmm2, XMMWORD PTR [rsp+16]
```

```
17      mov rax, QWORD PTR fs:40
18      mov QWORD PTR [rsp+88], rax
19      xor eax, eax
20      movapd  xmm3, XMMWORD PTR .LC5[rip]
21      mov eax, 6
22      mulpd   xmm3, xmm2
23      movsd   QWORD PTR [rsp+56], xmm6
24      movsd   QWORD PTR [rsp+40], xmm5
25      mulpd   xmm2, xmm7
26      movsd   QWORD PTR [rsp+64], xmm6
27      movsd   QWORD PTR [rsp+72], xmm6
28      movsd   xmm1, QWORD PTR .LC2[rip]
29      movsd   QWORD PTR [rsp+8], xmm6
30      movsd   QWORD PTR [rsp+32], xmm1
31      movupd  xmm5, XMMWORD PTR [rsp+48]
32      addpd   xmm5, xmm3
33      movapd  xmm3, xmm2
34      movupd  xmm2, XMMWORD PTR [rsp+32]
35      movupd  xmm4, XMMWORD PTR [rsp+64]
36      mulpd   xmm7, xmm2
37      addpd   xmm2, xmm2
38      addpd   xmm3, xmm4
39      movapd  xmm4, xmm7
40      addpd   xmm2, xmm3
41      movapd  xmm3, xmm6
42      addpd   xmm4, xmm5
43      movups  XMMWORD PTR [rsp+64], xmm2
44      movapd  xmm2, xmm1
45      movups  XMMWORD PTR [rsp+48], xmm4
46      movsd   xmm5, QWORD PTR [rsp+64]
47      movsd   xmm4, QWORD PTR [rsp+48]
48      call    __printf_chk
49      movsd   xmm6, QWORD PTR [rsp+8]
50      mov esi, OFFSET FLAT:.LC7
51      movsd   xmm5, QWORD PTR [rsp+72]
52      mov edi, 1
53      movsd   xmm4, QWORD PTR [rsp+56]
54      mov eax, 6
55      movsd   xmm3, QWORD PTR .LC1[rip]
56      movapd  xmm2, xmm6
57      movsd   xmm1, QWORD PTR [rsp+40]
```

```
58      movsd   xmm0, QWORD PTR [rsp+24]
```

3 Exercise 3: 书写 SIMD 代码

运行结果截图如下：

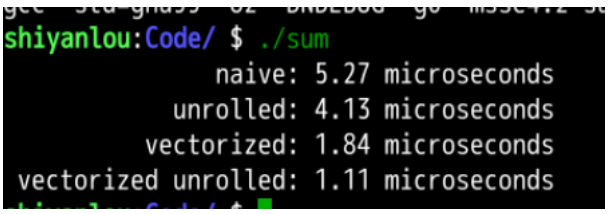

图 1: sum_vectorized

向量化相较于直接求和性能提升了越 2.8 倍

4 Exercise 4: Loop Unrolling 循环展开

运行结果截图如下：

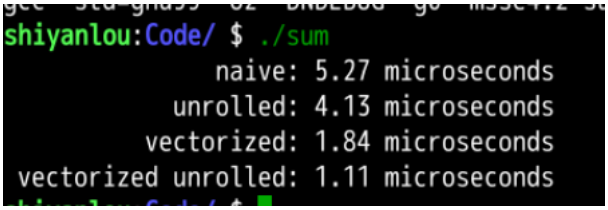

图 2: sum_unrolled

在向量化的基础上，循环展开比不展开性能提升了约 1.5 倍；向量化并循环展开比直接求和性能提升了将近 5 倍。

5 Exercise 5

对 sum.c 进行 O3 编译后得到的运行结果如图 3，可以看出程序会被编译器自动向量化，性能提升了大约 2.5 倍。

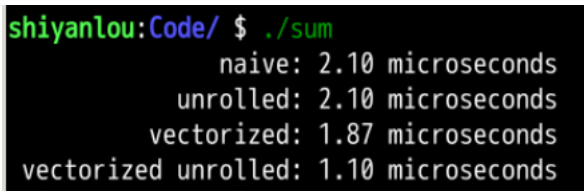

图 3

如果加上条件分支语句，结果如图 4向量自动化的效果减弱了。

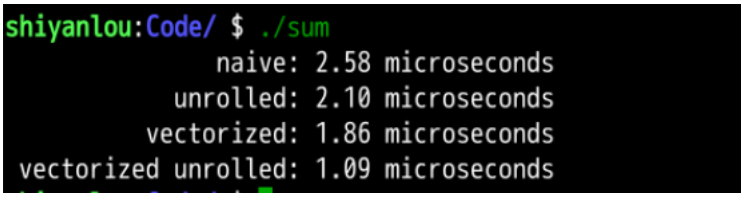

图 4:

使用如下编译指令对 sum256.c 进行编译：

```
gcc -O2 -mavx2 sum256.c -o sum256
```

得到结果如图 5，可以发现，当使用 256 位并行度的 intrinsic 函数时，性能提升相较于 128 位更加明显，相较于不用向量提升了 6.6 倍，提升的幅度降低了。这是由于当并行的数据增多时，影响运行速度的主要因素不再是数据的计算，而是数据的读取。因此即使是并行的数据增加一倍，性能提升也不可能随之增加一倍。

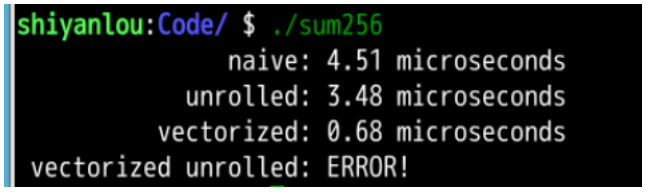

图 5: