实践4 实验报告

陈文迪 519021910071

Exercise 1: 熟悉 SIMD intrinsics 函数

问题回答：

- 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，有两种形式，后者为立即数

```
__m128i _mm_sra_epi16 (__m128i a, __m128i count)
```

```
__m128i _mm_srai_epi16 (__m128i a, int imm8)
```

Exercise 2: 阅读 SIMD 代码

```
movapd  xmm7, XMMWORD PTR 32[rsp]
addpd   xmm0, xmm0
mov QWORD PTR 72[rsp], 0x000000000
mulpd   xmm2, xmm6
mov QWORD PTR 80[rsp], 0x000000000
mulpd   xmm6, XMMWORD PTR 32[rsp]
mulpd   xmm7, XMMWORD PTR .LC5[rip]
mov QWORD PTR 88[rsp], 0x000000000
addpd   xmm7, XMMWORD PTR 64[rsp]
addpd   xmm6, XMMWORD PTR 80[rsp]
addpd   xmm7, xmm2
addpd   xmm6, xmm0
movapd  xmm2, xmm1
movapd  xmm0, xmm10
movapd  xmm5, xmm6
movapd  xmm4, xmm7
```

movapd（Move Aligned Packed Double-Precision Floating-Point Values），mulpd（Multiply Packed Double-Precision Floating-Point Values），addpd（Add Packed Double-Precision Floating-Point Values）指令均是执行SIMD操作的。

Exercise 3: 书写 SIMD 代码

我们可以按照以下代码实现：

```
static int sum_vectorized(int n, int *a)
{
    // WRITE YOUR VECTORIZED CODE HERE
    __m128i sum = _mm_setzero_si128();
    __m128i temp = _mm_setzero_si128();

    for (int i = 0; i < n / 4 * 4; i += 4)
    {
        temp = _mm_loadu_si128(a+i);
        sum = _mm_add_epi32(temp,sum);
    }

    int ans = 0;
    int *vector = malloc(sizeof(int)*4);
    _mm_storeu_si128(vector,sum);
    for(int i = 0;i<4;i++){
        ans+=vector[i];
    }
    free(vector);

    for (int i = n / 4 * 4; i < n; i++)
    {
        ans += a[i];
    }
    return ans;
}
```

运行结果：

```
andycwd@DESKTOP-ES9E36H:/mnt/c/Users/12779/Downloads/lab5_code$ ./sum
              naive: 6.92 microseconds
          unrolled: 4.70 microseconds
        vectorized: 1.96 microseconds
vectorized unrolled: ERROR!
```

可以看到，使用向量加法函数之后性能得到了改善。

Exercise 4：Loop Unrolling 循环展开

在练习3的基础上，我们将循环展开，实现如下：

```
static int sum_vectorized_unrolled(int n, int *a)
{
    // UNROLL YOUR VECTORIZED CODE HERE
    __m128i sum = _mm_setzero_si128();

    for (int i = 0; i < n / 16 * 16; i += 16)
    {
        sum = _mm_add_epi32(_mm_loadu_si128(a+i),sum);
        sum = _mm_add_epi32(_mm_loadu_si128(a+i+4),sum);
```

```
        sum = _mm_add_epi32(_mm_loadu_si128(a+i+8),sum);
        sum = _mm_add_epi32(_mm_loadu_si128(a+i+12),sum);
    }

    int ans = 0;
    int *vector = malloc(sizeof(int)*4);
    _mm_storeu_si128(vector,sum);
    for(int i = 0;i<4;i++){
        ans+=vector[i];
    }
    free(vector);

    for (int i = n / 16 * 16; i < n; i++)
    {
        ans += a[i];
    }
    return ans;
}
```

运行结果：

```
andycwd@DESKTOP-ES9E36H:/mnt/c/Users/12779/Downloads/lab5_code$ ./sum
               naive: 4.17 microseconds
            unrolled: 2.84 microseconds
          vectorized: 1.14 microseconds
 vectorized unrolled: 0.41 microseconds
```

可以看到，性能进一步提升。