计算机体系结构Lab5

Exercise 1: 熟悉 SIMD intrinsics 函数

- **Four floating point divisions in single precision**

  `__m128 _mm_div_ps (__m128 a, __m128 b)`
- **Sixteen max operations over unsigned 8-bit integers**

  `__m128i _mm_max_epu8 (__m128i a, __m128i b)`
- **Arithmetic shift right of eight signed 16-bit integers**

  `__m128i _mm_srli_epi16 (__m128i a, int imm8)`

Exercise 2: 阅读 SIMD 代码

观察 sseTest.s 文件的内容，哪些指令是执行 SIMD 操作的?
movapd，movsd，addpd，mulpd，unpckhpd

Exercise 3: 书写 SIMD 代码

`sum_vectorized` 函数代码如下：

```
static int sum_vectorized(int n, int *a)
{
    int sum = 0;
    int sum_vect[4];
    __m128i _sum = _mm_setzero_si128();
    for (int i = 0; i < n / 4 * 4; i += 4)
        _sum = _mm_add_epi32(_sum,_mm_loadu_si128((__m128i*)(a+i)));
    _mm_storeu_si128((__m128i*)sum_vect,_sum);
    for(int i = 0;i<4;i++)
        sum += sum_vect[i];
    for (int i = n / 4 * 4; i < n; i++)
        sum += a[i];
    return sum;
}
```

运行结果如下：

```
               naive: 3.72 microseconds
            unrolled: 2.97 microseconds
          vectorized: 1.50 microseconds
 vectorized unrolled: 0.92 microseconds
```

可以看出，相对 `unrolled` 有性能提升。

Exercise 4: Loop Unrolling 循环展开

`sum_vectorized_unrolled` 函数代码如下：

```
static int sum_vectorized_unrolled(int n, int *a)
{
    int sum = 0;
    int sum_vect[4];
    __m128i _sum = _mm_setzero_si128();
    for (int i = 0; i < n / 16 * 16; i += 16)
    {
        _sum = _mm_add_epi32(_sum,_mm_loadu_si128((__m128i*)(a+i)));
        _sum = _mm_add_epi32(_sum,_mm_loadu_si128((__m128i*)(a+i+4)));
        _sum = _mm_add_epi32(_sum,_mm_loadu_si128((__m128i*)(a+i+8)));
        _sum = _mm_add_epi32(_sum,_mm_loadu_si128((__m128i*)(a+i+12)));
    }
    _mm_storeu_si128((__m128i*)sum_vect,_sum);
    for(int i = 0;i<4;i++)
        sum += sum_vect[i];
    for(int i = n / 16 * 16; i < n; i++)
        sum += a[i];
    return sum;
}
```

运行结果见Exercise 3，性能相比 `sum_vectorized` 有进一步提升。